MOO Methods for Multidisciplinary Design Using Parallel Evolutionary Algorithms, Game Theory and Hierarchical Topology: Numerical aspects and Implementation of Model Test Cases (Part 2)

Luis F. Gonzalez*, D.S Lee† and J. Periaux††

* School of Engineering Systems, Queensland University of Technology (QUT) , Australian Research Centre for Aerospace Automation (ARCAA), Brisbane, Australia,
e-mail: felipe.gonzalez@qut.edu.au

† CIMNE/UPC, Barcelona, Spain
e-mail: ds.chris.lee @gmail.com

†† Scientific Computing Lab, University of Jyvaskyla, Finland
e-mail: jperiaux@gmail.com

1. SUMMARY

These lecture notes describe the use and implementation of a framework in which mathematical as well as engineering optimisation problems can be analysed. The foundations of the framework and algorithms described -Hierarchical Asynchronous Parallel Evolutionary Algorithms (HAPEAs) - lie upon traditional evolution strategies and incorporate the concepts of a multi-objective optimisation, hierarchical topology, asynchronous evaluation of candidate solutions , parallel computing and game strategies. In a step by step approach, the numerical implementation of EAs and HAPEAs for solving multi criteria optimisation problems is conducted providing the reader with the knowledge to reproduce these hand on training in his – her- academic or industrial environment.

2. INTRODUCTION AND MOTIVATION

Design and optimisation in aeronautics are complex tasks as non-linearities, multi-objective, multidisciplinary considerations are involved in the optimisation procedure. In order to handle this level of complexity it is desirable to develop a system, which facilitates integration of a series of design tools, graphical user interfaces, post-processing capabilities and others to solve the problem, such a system is termed a framework. This lecture focuses on the requirements, development and implementation of a framework that uses evolutionary techniques and a series of analysis tools in which different multidisciplinary and multi-objective problems in aeronautics can be analysed. The fundamental idea with this framework is to simplify the task of integration to the user so he/she can focus on the problem itself. The idea on the development of this framework is a generic system that can be easily developed, maintained and extended.

The lecture is mainly based on Reference 17 and is organized as follows: section 3 describes some of the requirements for a robust framework, section 4 describes the framework and its components, section 5 describes the implementation of the framework, section 6 describes the HAPEA algorithm, section 7 details the numerical implementation of HAPEA, section 8 describes and applies the methodology to mathematical and aeronautical design test problems and section 9 provides conclusions and avenues for future research.

3. REQUIREMENTS FOR A MULTI-OBJECTIVE MULTIDISCIPLINARY DESIGN OPTIMISATION (MDO) FRAMEWORK IN AERONAUTICS.

For a framework to be robust, practical and efficient it needs to satisfy a series of requirements, these can be subdivided in (i) problem formulation and (ii) optimisation methods, (iii) problem execution, (iv) architectural design and (v) information access [2-5,23,33,34,36,37].

(i) Problem Formulation

1. The framework should allow the user to configure and reconfigure different MO and MDO formulations easily without low level programming.
2. The framework should handle problems formulations that can be multi-objective, multi-modal, discontinuous, or with noisy search spaces.

(ii) Optimisation Methods.

3. The framework should allow ease of integration of robust optimisation methods.
4. It should also allow integrating different disciplinary analysis with different optimisation methods and should provide schemes which involve sub-optimisation within each design module.

(iii) Problem Execution.

1. The framework should allow the execution and movement of data in an automated fashion;
2. Should be able to execute multiple processes in parallel and through heterogeneous computers ;
3. Also a batch mode should be implemented.

(iv) Implementation and Architectural Design.

1. The framework should be developed using object-oriented principles;
2. The framework should allow the user to incorporate legacy codes, which can be written in different programming languages, and proprietary software where the source code is not available;
3. The framework should provide an easy to use and intuitive GUI;
4. The framework should be easily extended by developing new interfaces required to integrate new processes into the system;
5. The framework should not impose unreasonable overhead on the optimisation process;
6. The framework should be based on standards.

(v) Information Access.

1. The framework should provide facilities for database management;
2. The framework should provide capabilities to visualize intermediate and final result from the analysis or optimisation;
3. Also should allow capabilities for monitoring and viewing the status of an execution and its system status;
4. Also a mechanism for fault tolerance.

4. FRAMEWORK

With these requirements in mind the general scope for the framework was identified. The framework developed in this research address these requirements to some extent. Figure 1 shows a representation of different components to satisfy the requirements. The framework will have seven major constituents: a robust optimisation tool, a problem formulation capability within each analysis module, some architectural design considerations such as a GUI, a Design of Experiments (DOE) module (capabilities for LHS, orthogonal and RSM, Kriging), analysis modules, and capabilities for parallel computing and post-processing. In the following sections and subsection each of these constituents is detailed.

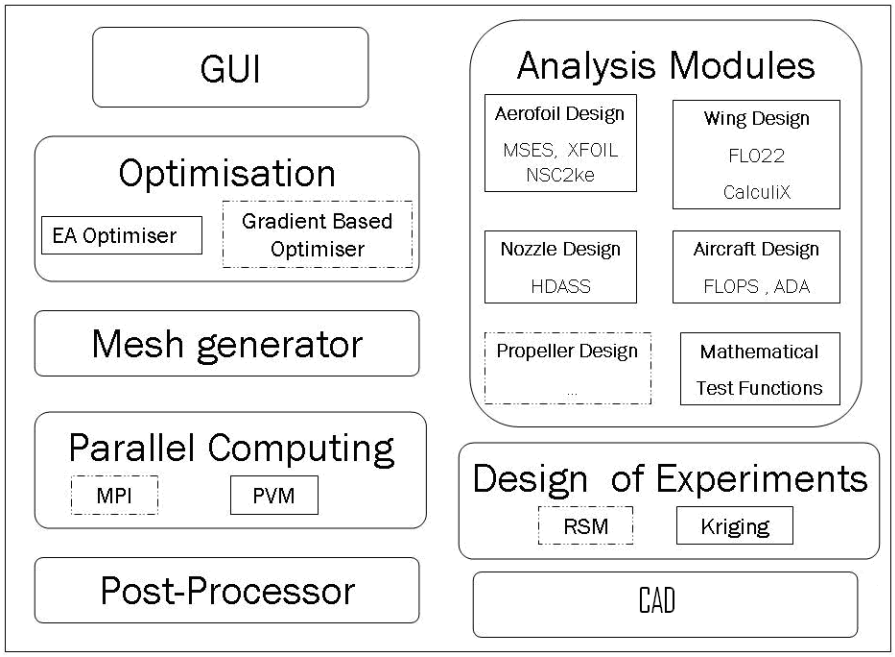


Figure 1: MDO Framework.

5. IMPLEMENTATION OF THE FRAMEWORK

Integrating all these components is a complex task. This work considers a general formulation for MDO and multi-criteria problems, the selection and description of a robust optimisation tool, the development of the architecture, the GUI and capabilities for pre and post-processing. The DOE capability has been accounted for, but has been evaluated only for simple mathematical test cases. The following sub-sections detail how the requirements are satisfied.

Problem Formulation and Solution

There are mathematical and engineering optimisation problems that give difficulties or cannot be solved by traditional gradient based techniques. When these problems are formulated it can happen that the search space is multi-modal, non-convex or discontinuous, with multiple local minima and noise, problems where we look for multiple solutions simultaneously, a Nash equilibrium point or a set of non-dominated solutions. Some problems in aeronautical and UAV design fall into this category.

Population based techniques are becoming popular as they can be suited to solve these type of problems. One major drawback of EAs is that they are slow in converging as they require a large number of function evaluations to find optimal solutions and have poor performance with increasing number of discipline specific and interdisciplinary variables. Hence the continuing effort has been on developing robust but faster numerical techniques to solve complex problem formulations, overcome these challenges and facilitate the complex task of design and optimisation in aeronautics.

Selection of a Robust Optimisation Tool

A second consideration is the selection of an optimisation tool that is appropriate for the problem to solve. For many problems traditional gradient based method, a canonical simple Genetic Algorithm (GA) or evolution strategy (ES) will be enough but more complex problems require a more robust approach. In this direction we use the Hierarchical Asynchronous Parallel Evolutionary Algorithm (HAPEA) described in section 6.

Architectural Design and Information Access

To satisfy the architectural design requirements the framework uses an object-oriented approach in C++. The benefits of using object-oriented software are the ease of implementation and extension of software in a modular fashion by the use of classes and methods. In an industrial and academic environment the need for a user-friendly application is required hence a simple GUI was designed. There were many considerations and options for the GUI development, but knowledge in C++ and the use of object-oriented principles were the main considerations. The Fast Toolkit (FLTk) library was selected for this task. This toolkit provides a friendly and easy to use environment for different implementations. The GUI is simple and modular on its implementation and consists of five main modules as illustrated in Figure 2. The main modules are: Design and Analysis, Design of Experiments, Post-processing and Parallel Processing. The GUI facilitates development, extension and modifications of modules in a rather simple manner. The user has to create only a few subroutines within the corresponding module.

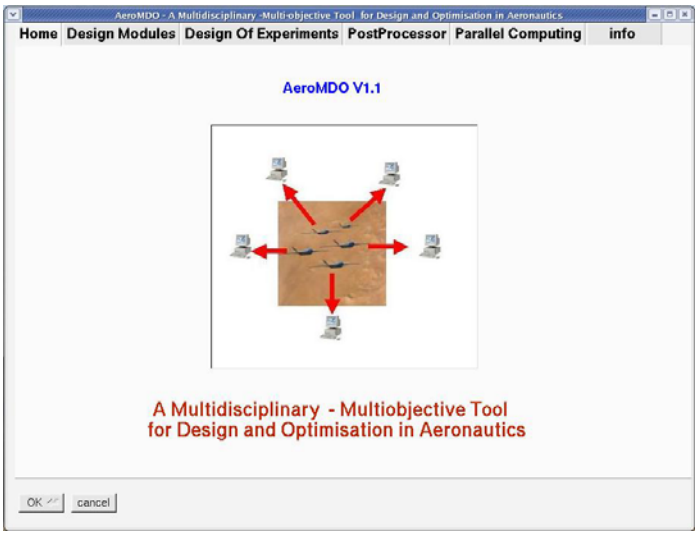


Figure 2: GUI Sample

Design and Analysis Module

A design module allows the user to conduct a single design and optimisation for different aeronautical applications and mathematical test cases. So far this module contains five sub-modules for aerofoil, multi-element aerofoil, nozzle, wing, aircraft and mathematical functions design or optimisation. As designed the framework is flexible and provides for ease of implementation of other design modules. Modules currently under development are such as those for propeller, cascade aerofoil and rotor blade design.

Development of Aeronautical Design Modules

Before implementing a sub-module it is necessary to develop a design module interface, this comprises a series of files written in C++ that allow communication between the GUI, analysis codes, the optimiser and the parallel processing capability. When designing the interface a choice has to be made depending if the source code for the analysis tool was available or not. In the current implementations minimal modification to the source code was required, ideally it is desirable to operate only through the input/output files of the analysis tool. In the implementations considered, a design template was used in conjunction with one or two additional files which contain the necessary linking subroutines allowing a rather fast implementation of the design modules. So far, there are subroutines for aircraft, nozzle, wing and full aircraft configuration

design. Each of these options allows the user to perform a single design analysis or a full optimisation. A general algorithm for the implementation of a new design module is represented in algorithm 1.

Aerofoil Design and Optimisation Module: This module allows the user to perform a single analysis or a full aerofoil optimisation routine. Three different CFD codes at a combination of them can be used: A panel method (XFOIL) [10], an Euler + boundary layer (MSES) [11] or Navier-Stokes analysis (NSC2ke [26]).

Wing Design and Optimisation Module: This module allows the user to conduct a single analysis on a wing or an optimisation study. These could be studies in one or several objectives or with multiple disciplines.

```
While Stopping condition not met;       // Infinite loop
    Receive information from optimizer; design variables, constraints
    Generate Geometry (i.e. aerofoil, nozzle, wing)
    for i=0, n    // n: Number of objectives
        Evaluate candidate geometry using analysis tool (i.e.  CFD-FEA)
        Check constraint violation
        Calculate fitness
       Return the computed individual (Design variables + fitness vector to optimiser)
end loop
```

Algorithm 1: Design modules algorithm.

Aircraft Design and Optimisation Module: This module allows the user to analyse and optimise different problems related to aircraft external configuration design. It can be used to design and optimise different subsonic, Unmanned Aerial Vehicles, transport or supersonic aircraft. Single or multi-criteria optimisation studies can be performed. Comparison of different multi-criteria analysis such as Pareto optimality and Nash equilibrium approach are possible. The user can select from two different analysis codes: An object-oriented Aircraft Design and Analysis Software (ADA) developed by the first author or using the Flight Optimisation System (FLOPS) software developed by A. McCullers at NASA Langley. ADA is conceptual design and analysis software written using object-oriented principles and is based on the formulation described in Raymer [32]. FLOPS [24], a more robust solver, is a workstation-based code which has capabilities for conceptual and preliminary design and evaluation of advanced concepts. The sizing and synthesis analysis in FLOPS are multidisciplinary in nature. It has a numerous modules and analysis capabilities for takeoff, performance, structural, control, aerodynamic and noise. This code is used in some universities as well as aerospace firms and government for MDO development and it allows an integral multidisciplinary analysis for the entire aircraft mission and calculation of performance parameters such as range, endurance, takeoff field length and landing field length.

Multi-element Aerofoil Design and Optimisation Module: Similar to the aerofoil design module, it allows the user to perform a single analysis or a full optimisation, the user can choose from an Euler or Navier-Stokes analysis.

Mathematical Test Functions Module: This module allows the user to design, and evaluate single, or multi-criteria mathematical test functions which give confidence in the robustness and performance of the optimisation method before deciding on its application to real world problems. The current implementation includes mathematical test function for single or multiple criteria, constrained optimisation, DOE and non-linear goal programming problems.

Design of Experiments Module
In the implementation considered in this research, the designer uses an EA for the optimisation, but as discussed in section 1, one of the drawbacks of EAs is that they suffer from slow convergence. By providing a DOE capability into the framework we wish to hybridize the desirable characteristics of EAs and surrogate models such as RSM to obtain an efficient optimisation system. Within this context, the DOE samples a number of design candidates at which the analysis code (CFD will run), the surrogate model is then constructed for the computationally expensive problem. Different sampling and DOE strategies can be used; Latin hypercube, Response Surface Methods or Kriging. If desired, the user can design and implement different DOE methods.

Parallel Computing Module
One of the drawbacks of EAs is slow convergence but this module allows the users to dynamically create, add or delete nodes on the parallel implementation. Recent work on multi-criteria parallel evolutionary algorithms has allowed significant performance and robustness gains in global and parallel optimisation [6, 40]. The framework considers the implementation of a cluster of PCs, wherein the master carries on the optimisation process while remote nodes compute the solver code. The message-passing model used is the Parallel Virtual Machine (PVM) [12].

Post Processing
The approach considered for post-processing was to use a combination of visualisation capabilities within each analysis software, and the use of GNU plot (a graphics software common in most UNIX installations). Common to all design modules is visualisation of the evolution progress of the fitness function and Pareto fronts for multi-criteria problems. Post-processing tools on each analysis module include a top view of the wing plan forms and a general 3D view of the resulting aircraft configurations. Visualisation tools within each analysis software module include the pressure coefficient distribution on the aerofoil using an Euler/Boundary layer solver or pressure or Mach contours using a Navier-Stokes solver.

Implementation of Different Legacy Codes
The framework also implements legacy codes in different programming languages C, C++, Fortran 90, and Fortran 77. The optimiser has been successfully coupled with the following aerodynamic and analysis software: FLO22 [20] FLOPS, ADA, XFOIL, MSES, and NSC2Ke. One of the benefits of using an Evolutionary optimiser is that EAs require no derivatives of the objective function. The coupling of the algorithm with different analysis codes is by simple function calls and input and output data files.

6. DESCRIPTION OF HIERARCHICAL ASYNCHRONOUS PARALLEL EVOLUTIONARY ALGORITHM.

As described in section 5.2, the core of the framework is based on a Hierarchical Asynchronous Parallel Evolutionary Algorithm (HAPEA). The foundation of this algorithm lie on traditional evolution strategies and incorporate the concepts of multi-criteria optimisation, hierarchical topology, parallel computing, asynchronous evaluation and game theory (Pareto tournament, Nash Approach).

6.1 Evolution Strategies

Preliminaries
As described in the previous paper, the canonical ES that we consider uses multi-membered populations for both parents and offspring, and has two basic forms; the (μ+ λ)--ES and the (μ, λ)--ES. These both apply a variation operator to the parent population μ to generate an offspring population λ [25].

The selection method is either the elitist plus operator (+) where the next parent population is found by determining the μ best individuals from the union of the parent and offspring sets, or the non-elitist comma operator (,) here the next parent population is found from the μ best individuals of λ only. These are often referred to in the literature as simply 'the plus strategy' or 'the comma strategy'.

It is also necessary to introduce quantities which are handled by the Evolution Strategy so that we may define the algorithm itself. Firstly it is assumed without loss of generality that the given problem is one of minimisation:

$$\mathbf{f}^* = min(f(x)) \tag{1}$$

Where **f*** is the minimum possible value of the fitness function (a global minima) for all admissible values of the problem variables (also referred to as object variables). All problems may be cast in this form, even if they are more 'naturally' suited to a maximisation (such as range or payload for aircraft), by a simple mathematical manipulation such as:

$$f_{min} = 1/f_{max} + \varepsilon \tag{2}$$

Where ε may be selected non-zero if there is a possibility that $\mathbf{f}_{max}=0$

We define an individual as an object containing a vector of problem variables (**x**), a vector of strategy variances (σ) and rotation angles (α):

$$I=(\mathbf{x}, \sigma, \alpha)=((x_1, x_2, \ldots, x_N),(\sigma_1, \sigma_2, \ldots, \sigma_N),(\alpha_1, \alpha_2, \ldots, \alpha_{N(N-1)/2})) \tag{3}$$

Where N is the number of variables assigned to the problem (the problem dimension). Strategy variables were explained in the previous paper. We assume here the number of strategy variances is equal to the problem dimension, but other possibilities exist where the number of variances may be fewer (but never larger).

Overall Algorithm

The overall operation of the algorithm is merely the repetition of a generational loop. This is the application of the recombination operator, the mutation operator, the fitness function and the selection operator. A pseudo code of a canonical evolution strategy is illustrated in algorithm 2. A population (μ_o) is initialised and then evaluated. Then for a number of generations (g) and while a stopping condition (maximum number of function evaluation or target fitness value) is not met, offsprings (λ^{g+1}) go recursively through the process of recombination, mutation, evaluation and selection.

Initialise: $init(\mu^0)$
Evaluate: $f(\mu^0))$
$g=0$
while stopping condition not met,
  Recombine: $\lambda_R^{g+1} = reco(\mu^g)$
  Mutate: $\lambda_M^{g+1} = mut(\lambda_R^{g+1})$
  Evaluate: $\lambda^{g+1} = f(\lambda_M^{g+1})$
  Select: $\mu^{g+1} = sel(\mu \cup \lambda)$ (plus strategy) or,
  $\mu^{g+1} = sel(\lambda)$ (comma strategy)
$g=g+1$
loop

Algorithm 2: Canonical Evolution Strategy.

6.2 Modern Techniques and Extensions

Even though the canonical evolution strategy is a good approach for solving some problems [18] it had to be extended to have an algorithm that:

- Utilises a cheap, readily available parallel processing capability running variable-time iterative solvers on desktop computers, through *asynchronous solution* ;
- Allows for the exploitation of variable-fidelity or multi-physics solvers, through a *hierarchical population topology* ;
- Improves convergence speed and increase the likelihood of finding a solution, through the coupling of the covariance *matrix adaptation;*.
- Can be applied to more varied types of engineering problems in one or many objectives, through *Pareto tournament selection***.**

All the extensions are fully integrated into a single optimiser code, capable of finding solutions to problems in the situations given above.

6.2.1 Asynchronous Solution

The first extension is a parallel computing capability described in the previous notes. It has to be noted that most evolutionary algorithms are configured suing a synchronous approach but this as some drawbacks. In this work we extend the concept of ESs and parallel computing by implementing the asynchronous approach. The design of an optimisation algorithm for asynchronous solution of candidate problems can be broadly defined as the non-dependence of an optimisation procedure on the speed of a given solver [39]. The need for asynchronous fitness function evaluation arises from the fact that many methods of solution used in engineering today may take variable times to complete their operation. The classic example of this is the modern CFD solver. In a typical industrial code used for external aerodynamic analysis of aeroplanes, the time for the residual of the solution to converge to a specified level (either machine zero or an arbitrarily selected higher value) can vary over a significant range. If as an example, we are given an unstructured mesh based Navier-Stokes solver, a number of factors have significant effect:

- The number of cells involved, and the simplicity of their connectivity.
- The skewness and aspect ratio of the cells, especially considering we are forced to use automatic mesh generation.
- The presence of unforeseen large gradients in the flow such as confluent boundary layers, merged shocks and flow separation.
- The degree of mesh anisotropy in the area of large flow gradients, which will probably have to be automatically adapted.

With all of these factors considered, it should seem obvious that the optimisation algorithm used to drive the design process should be designed with this possibility in mind. Other examples of computational analyses that may take variable amounts of time are nonlinear structural FEA (buckling and vibration) and electromagnetic influence analysis (radar backscattering). In fact, with any analysis method based upon the iterative integration of sets of nonlinear partial differential equations (PDEs), it must be assumed that the time taken for each different configuration run will be different. Even with linear tools such as linear structural FEA used for static strength analysis, where the solution time can be accurately approximated *a priori*, the fact that automatic internal mesh generation will still be used influences the time taken to arrive at a solution.

The previous generation of evolutionary algorithms have generally used a generation based approach. The canonical genetic algorithm and evolution strategy both use this approach. A problem with generational models is that they create an unnecessary *bottleneck* when used on parallel computers. If the population size is approximately equal to the number of processors, and most candidate offspring sent for solution can be successfully evaluated, then some processors will complete their task quickly with the remainder taking more time. With a generational approach, those processors that have already completed their solutions will remain idle until all processors have completed their work.
A final and most important need for asynchronous evaluation is that it provides an ideal method of using existing desktop computers for problem solving. The need for asynchronicity arises due to the fact that many of these machines will have different operating speeds, and may be added or removed from the parallel task when they are needed for other work; meaning that no correct number or combination of resources can be known in advance.

The approach used in this work [42], is to ignore any concept of generation based solution. This approach is similar to work done by Wakunda and Zell [41], however the selection operator is radically different (please refer to Pareto Tournament selection description further in this paper), so it does not utilise the truncation selection operator. Whilst a parent population exists, offspring are not sent as a complete 'block' to the parallel slaves for solution. Instead one candidate is generated at a time, and is sent to any idle processor where it is evaluated at its own speed. When candidates have been evaluated, they are returned to the optimiser and either accepted by insertion into the main population or rejected. This requires a new selection operator because the offspring cannot now be compared *one against the other*, which is the standard selection technique used in generational algorithms. In fact, a single offspring must compete against a previously established benchmark and if successful must replace (according to some rule) an individual pre-existing in the population.

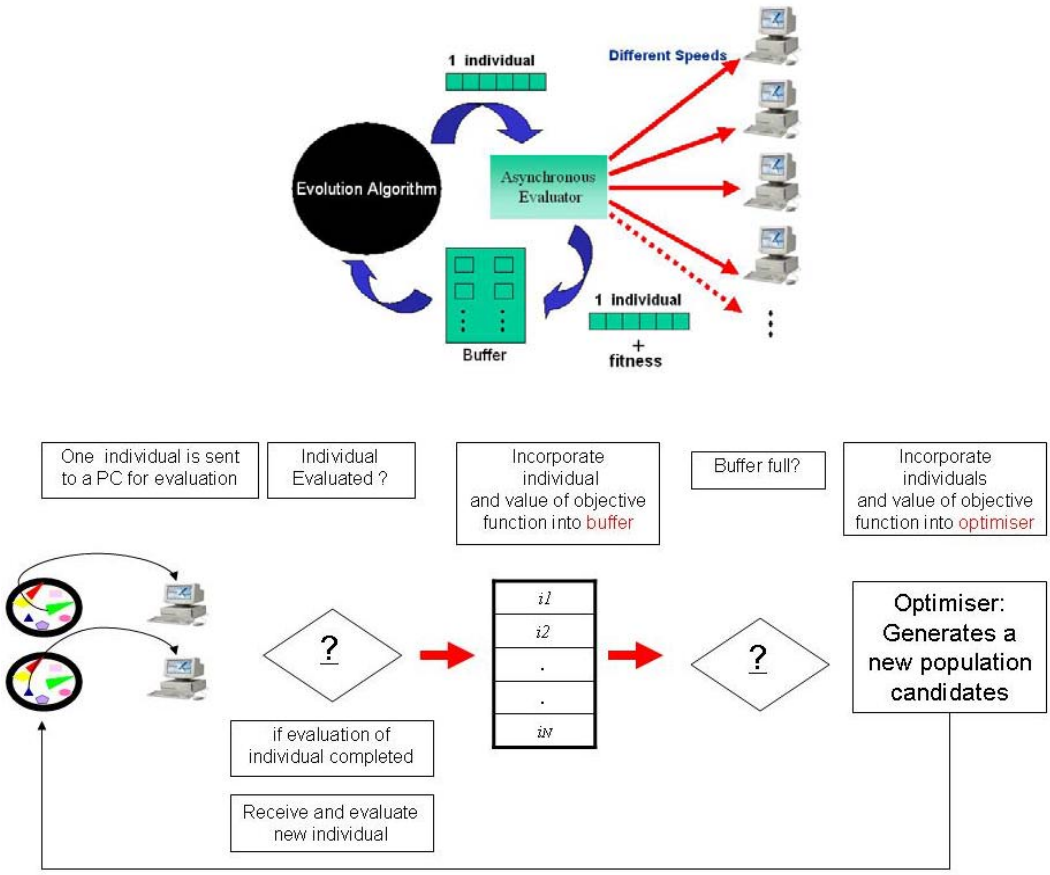


Figure 3: Parallel Computing and Asynchronous Evaluation of individuals.

We implement this benchmarking via a separate evaluation buffer, which provides a statistical 'background check' on the comparative fitness of the solution. The length of the buffer should represent a reasonable statistical sample size, but need not be too large; approximately twice the population size is more than ample. The process is illustrated in figure 3. When an individual has had a fitness assigned, it is then compared to past individuals (both accepted and rejected) to determine whether or not it should be inserted into the main population. If it is to be accepted, then some acceptance rule (or *replacement strategy*) is invoked and it replaces a member of the main population μ. Some possible replacement strategies are:

Replace--Worst--Always: This strategy is a good choice for most situations. It is a compromise strategy, providing for some elitism in μ by allowing the superior individuals to persist. On the other hand, it ensures good information continues to flow into the population by always being invoked.

Replace--Oldest--Always: This is another good selection, and the closest operator to the canonical ES. It ensures continuous information flows into the population, and any individual can be replaced regardless of its superiority -- age is the only factor. Also, this is the closest model to natural species.

Replace--At--Random: This is a possible strategy which allows no bias whatsoever in the selection of which individual to replace. While completely egalitarian, the performance of the algorithm has been found to be inferior on all test cases, so we will not consider it further.

Replace--Worst--If--Better: This is an elitist strategy whereby the worst individual is only replaced if a superior one is found. While this guarantees a monotonic improvement of population fitness, it does also inhibit rapid information flow and dynamic exploration.

Replace--Oldest--If--Better: This is similar to *replace--worst--if—better*, merely replacing the oldest member if a better one is found. This also is a particularly stringent operator and should only be used in situations where the fitness function is known to be smooth and uni modal; or nearly so.

The importance of an algorithm which is not strictly elitist cannot be overlooked. The effect of continuous selection information -- both good and bad -- ensures correct updating of the strategy parameters and this effect has been studied by experts in the field of Evolution Strategies [25]

In summary, the important change required of an evolution algorithm to implement asynchronous evaluation is the splitting of the variation operator and the selection operator, and the provision of a sound basis for evaluating the fitness of solutions generated 'on the fly', given that in general the main population itself will be too small for this purpose. In the software developed for this these, the parallel implementation is performed using the open source Parallel Virtual Machine (PVM) framework which offers simple data sharing and broadcast facilities over heterogeneous clusters of computers.

A Short Simulation

To demonstrate the effectiveness of the asynchronous evaluation method, we give a short simulation. In this simulation, we configure the evolution algorithm in a conventional way (using a synchronous approach). We solve the simple sphere function f_1 [24], on a single computer using two methods of evaluation:

Asynchronous:

Assign a small fictitious delay to each function evaluation. This will vary uniformly between two values, fastest to slowest. Evaluate the individuals asynchronously.

Synchronous:

Assign the same delay to all individuals in advance. However, wait until the slowest evaluation has completed -- in exactly the same manner that this would occur in practice over a cluster of computers.

The purpose for entering into a simulated demonstration is that the computational assembling statistical data for this type of experiment using a real test case is presently prohibitive. We run the simulation using four unknowns (N=4) until a stopping condition of $f_{best} < 10^{-4}$ is reached. Each point is run 25 times and the average execution time is compiled. The result is presented for both asynchronous and synchronous runs in figure 4. The results are scaled vertically so that when no synchronicity exists ($t_{slowest}/t_{fastest}=1$), the workload factor is also unity; in any case it can be seen that both algorithms perform identically in this situation just as expected. Configurations up to $t_{slowest}/t_{fastest} =5$ are explored. It is obvious from figure 4 that the synchronous approach fares poorly when a higher variability exists in execution time, when compared to the asynchronous approach.

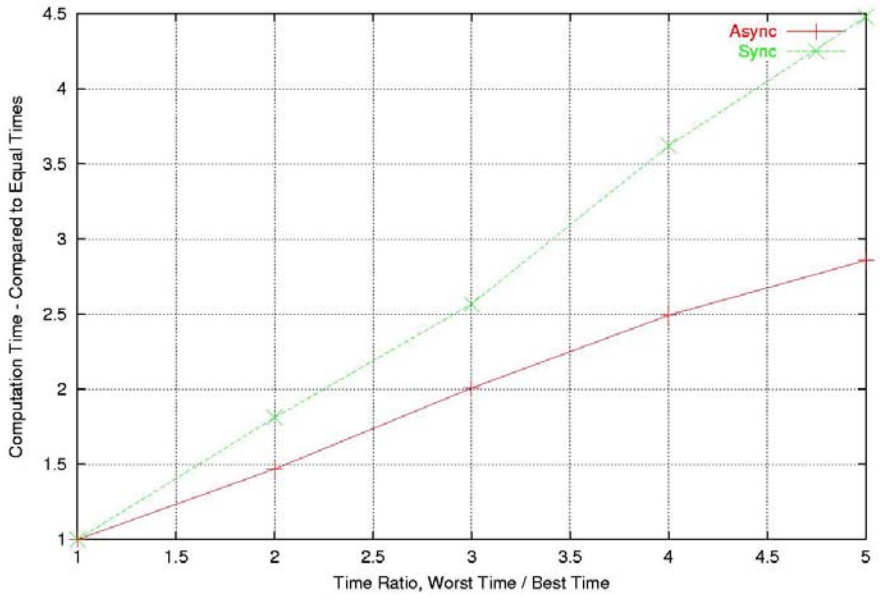


Figure 4: Execution time variation between asynchronous and synchronous evaluation.

6.2.2 Hierarchical Population Topology –multi-fidelity multi-populations approach

The second extension to the canonical evolution strategy is a hierarchical topology -multi-fidelity multi-populations approach described in the previous paper and detailed in the following. A hierarchical population topology, when integrated into an evolution algorithm, means that a number of separate populations are established in a hierarchical layout to solve the given problem, rather than a single 'cure-all' type single population layout. This method was first proposed by Sefrioui [35], and is shown in figure 5.

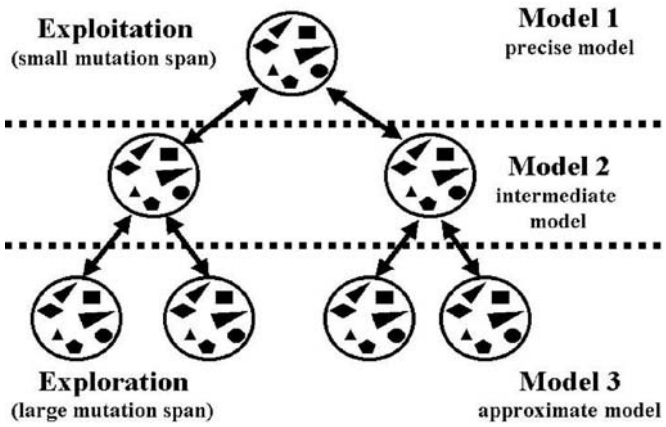


Figure 5: Hierarchical Population Topology.

The purpose of utilising a hierarchical topology is to exploit the possibility of using incomplete or rapid function evaluation information during optimisation progression. In other words, perfect (or the most compete available) evaluation of the fitness function is not required all the time in order to solve a given problem. The advantage of the hierarchical method is predominantly speed; to achieve a given quality level for the solution a much shorter wall--clock time is required.

Implementation and Advantages

In any multiple population method, there are a few additional issues that need to be addressed when laying--out the algorithm:

- How many individuals are placed in each population (equal or unequal, few or many);
- How the populations are connected to each other for migration (circular, torroidal, box or hierarchical);
- What period should be given to the isolation phase (time, number of evaluations or improvement in fitness)
- Which individuals should migrate (randomly selected or best few).

The hierarchical methodology provides the connectivity for the layout; there is a head population (or *trunk*), followed by successive layers of sub-populations (or *leaves*). Throughout this work we will use the binary tree configuration whereby each population in a layer is connected to two populations in a lower layer, and individuals are passed in both directions (i.e. up and down) between the layers. The most common layout is usually three layers corresponding to one, two and four populations (for a total of seven), however the method is extensible to any number of layers, and lower layers need not necessarily connect in a binary fashion to the upper layers.

The advantage of the hierarchical layout is that it appears quite suitable for engineering type problems, where the fitness function can be evaluated with varying levels of precision. Note that this concept is in keeping with the theme developed in the previous subsection -asynchronous approach- and it is hoped that the two techniques applied together produce a significant speedup of the optimisation process. The layers are arranged so that the uppermost layer uses the most precise solver available for fitness function optimisation. The next layer (or layers) are assigned intermediate quality solvers, and the lower-most layer uses the least accurate solver available. There are a number of ways of assigning this solver workload, but at the moment it is still operator dependent. The task of the lower layers is predominantly exploration, where large jumps

in design variables are expected and the population is not too particular about small improvements in fitness. The middle layer is a compromise position between the top and bottom layers. The top layer is tasked with *exploitation*, using very small steps in the search space and fine-tuning the final result. Provided that the solver on the lower layers is 'representative' of the problem, then the algorithm should proceed well. An exact definition of 'representative' is not possible as it is highly problem-dependent, however from experience, good rules of thumb for solvers on lower layers are:

- They need only contain the 'large-scale' physics of the next higher layer,
- They should be at least twice as fast as the next higher layer,
- They should produce broadly similar 'trend' information after the perturbation of a design variable; In other words, actual values of fitness are irrelevant provided all layers change the fitness value in the approximately the same direction after the same perturbation of a design variable.

As an example we can take the aerodynamic optimisation of a transonic aircraft wing. A first attempt at dividing the workload into three layers might proceed as follows

- Use a full compressible Navier--Stokes solver with turbulence modelling on the top layer;
- Use an Euler solver with coupled boundary layer on the intermediate layer;
- Use a full potential or Euler solver on the lowest layer.

This follows our rules of thumb where in this case the top layer is at least an order of magnitude slower than the lower layers. The full potential solver provides initial guesses at good solutions to the Euler layer, where they are rechecked and further evolved. Good solutions from the Euler layer progress up to the top level where they are again checked and further evolved. In this manner no time is wasted using the Navier--Stokes solver to explore large parts of the search space, which would be almost impossible in any reasonable time.

However, in software vendor environment access to many different types of solvers, incorporating different physical models like the ones above may not be available or allowed. As a second example, assuming that only one solver is available and it is of the Euler type, and includes automatic mesh generation, we could proceed as follows:

- Use a very fine mesh on the upper layer, and allow the solver residual to fall to effectively zero for every individual tested.
- Use a compromise mesh on the middle layer, and only let the solver residual to fall many orders of magnitude.
- Use a coarse mesh on the lowest layer, and only run the solver until the residual falls just a few orders of magnitude.

Again, this layout meets with our rules of thumb given above.

The concept is in keeping with another form of mathematical solution finding, namely the multigrid technique applied to partial differential equations [29]. The concepts are very similar; incomplete information from a lower quality level is passed to a higher quality level, significantly shortening the search time.
Only one question remains which is the implementation of the hierarchical method in an asynchronous context. We adopt the 'hands--off' approach throughout this work, whereby the migration phase is entered only when each of the populations is ready for migration. To this effect, we implement it as follows:

- Allow each population to run at its own speed. After a certain number of function evaluations have been completed by the population, it broadcasts that it is ready for migration. On a trial and error basis, it has been determined that an acceptable choice for the number of function evaluations before migration is approximately 2μ, and we adopt this throughout this work.
- Only when every population has broadcast that it is ready for migration do we enter the migration phase.
- Migrate *up* using the best third (1/3 μ) of the members of the population. This figure was determined by trial and error.
- Migrate *down* using a random third (1/3 μ) of the members of the population. Again, this was determined as broadly acceptable by trial and error.

6.2.3 New Mutation Operators

Covariance Matrix Adaptation (CMA)

The third extension to the canonical Evolution Strategy is the incorporation of the Covariance Matrix Adaptation (CMA) mutation model developed and extensively studied by Hansen and Ostermeier [18]. This mutation model takes as its inspiration conventional optimisation techniques, which work by developing a second order model of the fitness function. Through information gathered about the fitness landscape, learned through the selection process, a model of the covariance matrix is built up progressively. This then acts as a scaling metric on subsequent mutations. It also has the benefit of being derandomised; the same mutation vector is applied to both object and strategy variables, and this has been shown to significantly accelerate the optimisation process. Hansen has shown that the original mutation mechanism used by the canonical ES (section 6.1) is flawed, and the performance of this ES can depend on the permutation of the object variables -- clearly an undesirable situation. The basis of the method is that individuals no longer 'carry' the strategy variables themselves, and these are stored as a single set by the evolution algorithm itself [18]. Details on the CMA method implementation will be omitted here but ca be found in Reference 17.

Game Strategies

The next extension of the method is to implement capabilities within the optimisation method to be equally applicable to both single and different multi-objective (game theory) problems. Two concepts from Game strategies were implemented with the solver; the Pareto Tournament Selection and Nash Equilibrium [30].

Pareto Tournament Selection

A suitable selection operator capable of handling single or multi-objective problems was developed for the first concept. We implement the on-the-fly selection operator by means of a Pareto tournament selection operator. To implement an optimisation algorithm that is equally applicable to both single and multi-objective problems, a suitable selection operator capable of handling either situation must be developed. We propose an extension of the standard tournament operator popular in many approaches [8, 13, and 25]

Most evolutionary algorithms configured for multi-objective optimisation currently use the non-dominated sorting approach. This is a straightforward way to adapt an algorithm that is designed as a single objective optimiser into a multi-objective optimiser, and is used by many researchers [7, 9]. The problem with sorting approaches is that the method is not a fully integrated one. Briefly, a sorting method works by computing the set of non-dominated solutions amongst a large statistical sampling (either a large population or previous data), and assigning these solutions a

rank one. Then ignoring these points, the process is repeated until a 'second' Pareto front is found, and this is assigned rank two. This process continues until all points are ranked, and then the value of the rank is assigned to the individual as a new single objective fitness. An example of the ranking process is shown in figure 6.

A problem arises now on whether it is fair to assign individuals in the second rank numerically half the fitness of the first, and whether the third rank deserves a third of the fitness of the first. This poses a dilemma regarding the level of equality present amongst the solutions, as often solutions with excellent information may lie adjacent to, but not in, rank one. To solve this 'artificial scaling' problem, it is possible to introduce scaling, sharing and niching schemes, however all of these require problem-specific parameters or knowledge, even in adaptive approaches. It is of course always desirable to compose an algorithm that does not introduce such unnecessary parameters.

The current operator is a novel approach in that it requires no additional 'tuning' parameters, works seamlessly with the asynchronous selection buffer (B), and is very easy to encode. Simply, to determine whether a new individual x is to be accepted into the main population, we compare it with the selection buffer by assembling a small subset of the buffer called the t tournament functions $Q = [q_1, q_2, ..., q_n]$. We assemble Q by selecting individuals from the buffer, exclusively at random, until it is full. We then simply ensure that the new individual is not dominated by any in the tournament. If this is the case, then it is immediately accepted, and is inserted according to the replacement rules. The only parameter that needs to be determined in advance is the tournament size, a parameter that would exist in a single objective optimisation anyway. Selection of this parameter requires a small amount of problem specific knowledge, and should vary between $Q = \frac{1}{2} B$ (strong selective pressure) and $Q = \frac{1}{6} B$ (weak selective pressure). The optimiser is not overly sensitive to this value, provided the user errs on the side of weak selective pressure (smaller tournaments) in the absence of better information. The egalitarian approach to the tournament (by selecting individuals at random) ensures good diversity amongst the selected individuals; no niching or forced separation of individuals has been found necessary. It can also be seen that in the event the fitness vectors have only one element (a single objective optimisation), this operator simplifies to the standard tournament selection operator [43]. These concepts are illustrated in figure 7.

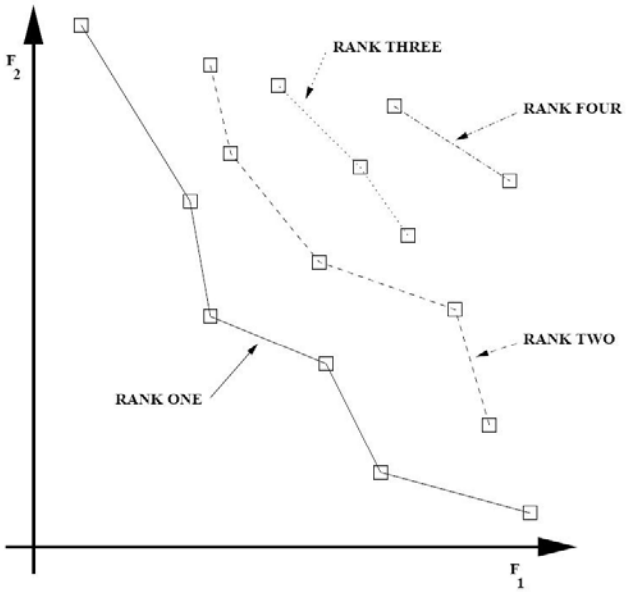


Figure 6: Pareto ranking process.

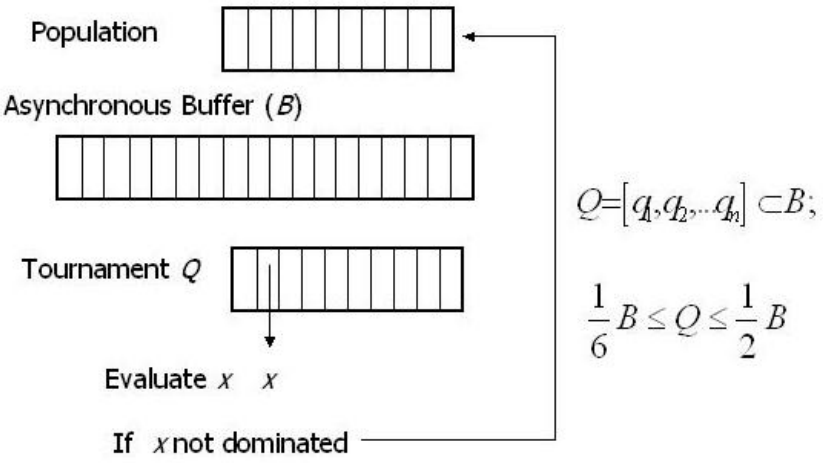


Figure 7: Pareto tournament selection operation.

Nash Solutions with HAPEA

The second concept on game strategies implemented within the solver is the Nash solutions approach described in the previous paper and further detailed here when it is combined with the HAPEA approach. The Nash equilibrium is determined by *n* players competing symmetrically for *n* criteria, where each player optimises a unique subset of optimisation variables, and all other variables are determined by the other players. For example, for player *i* the vector of problem variables is:

$$X = (x_1, \bar{x}_2, x_3, \bar{x}_4, \bar{x}_5, \ldots, x_N) \tag{4}$$

Where all variables x_i are free and the remainder are 'locked' by the other players. Player *i* is interested only in the objective, namely $f_i = f_i(X)$ where $F(X) = (f_1(X), f_2(X), \ldots f_n(X))$ is the entire multi-objective problem. We implement this using one EA for each player, as can be seen in figure 8, whereby information is exchanged between the EAs after a migration period has occurred.

There are two migrations present when using the hierarchical EA-Nash scheme, first there is a circulation of solutions up and down, the best solutions progress from the bottom layer to the top layer where they are refined and then a Nash migration where information between players is exchanged after an epoch the new variables for each player are updated on each node on each hierarchical tree.

7. DETAILS ON NUMERICAL IMPLEMENTATION OF THE ALGORITHMS

This section will describe in detail the implementation of the evolutionary algorithm. The overall operation of the algorithm is described initially, followed by a detailed description of its component parts. The implementation details of the concepts described in section 6 and the notes of the previous lecture are now given in detail.

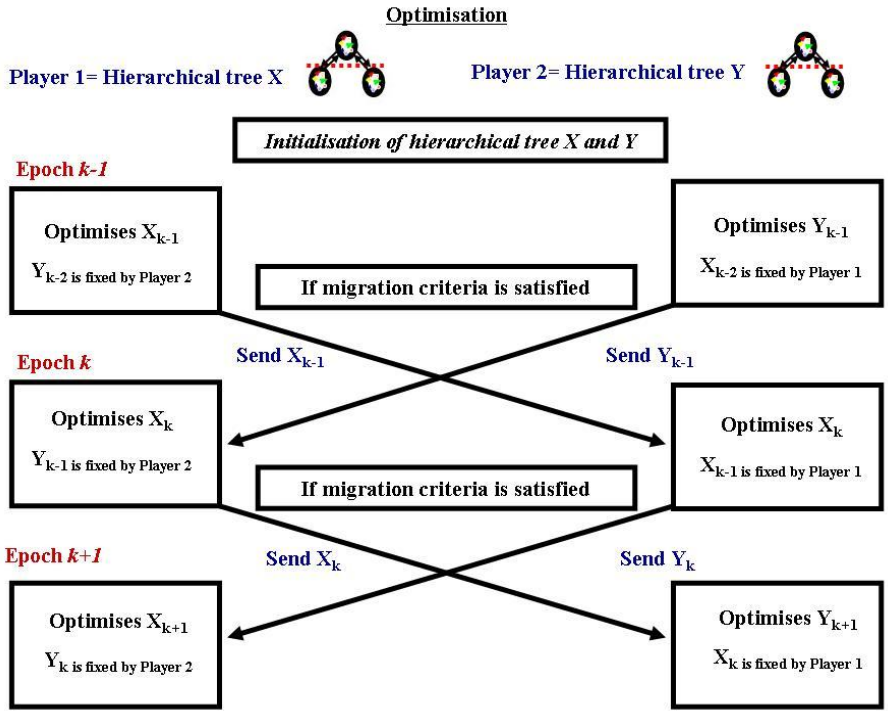


Figure 8: Nash Solutions with HAPEA.

7.1 Overall Operation

The overall operation of the algorithm is contained within essentially three modules, which are the fitness function (or solver), the routine handling returning individuals (*incorporation* routine) and the routine handling outgoing individuals (*generation* routine). The flow of information between these modules is continuously updated via an overall scheduling routine (*scheduler* routine), which represents the overall controller of the algorithm. The role of the scheduler is to split the generation and *incorporation* of individuals into two distinct steps, which then facilitates asynchronous operation of the algorithm as described in section 6.2.1 (the layout is shown in figure 9). The scheduler operates in a continuous loop and performs four major tasks. The scheduler first determines whether given stopping conditions (see section 7.4) have been met, and if so the evolutionary loop is exited and the entire process is stopped. If no stopping conditions are met, the scheduler updates the asynchronous solver (see section 6.2.1) so that further progress may be made. Then the scheduler determines whether or not candidate solutions which have been solved are ready for *incorporation* into the population. If such solutions exist, the *incorporation* routine (see section 7.2) is called and available candidates which now have had a fitness assigned are processed. Finally, the scheduler then determines whether it is possible to generate more candidates, by polling the asynchronous solver. If this is possible, then the generation routine (see section 7.3) is called and individuals are generated via the evolutionary operators. This operation is described in algorithm 3.

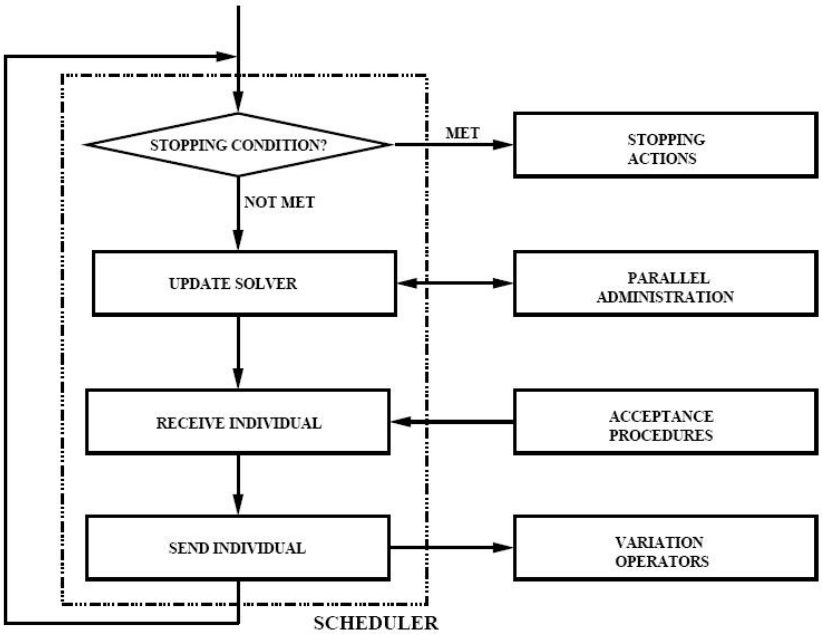


Figure 9: Overview of the scheduler process.

while stopping condition not met,
  Update the asynchronous solver.
    if solved individuals are available: incorporate them.
    if new individuals can be provided: generate them.
loop

Algorithm 3: Scheduler routine.

7.2 Incorporation of Individuals

After an individual has been evaluated it is returned to the incorporation routine. This routine includes the selection operator, the updating of selection information, the updating of other internal parameters as well as performing general housekeeping on the main population. The process is defined in algorithm 4. For simplicity components of this algorithm are described separately in algorithms 4, 5 and 6. Figure 10 gives a general overview of the process.

A new individual is received: I_{new}
Age population: For each μ_i in μ: $\mu'_{i\_age} = \mu_{i\_age} + 1$
Age selection buffer: For each b_i in B; $b'_{i\_age} = b_{i\_age} + 1$
Perform tournament selection: $accept = tournament\ (I_{new}, b)$
Delete oldest b_i in selection buffer, insert I_{new}
if $accept = true$
  Perform replacement in μ: $replace\ (\mu, I_{new})$
  Sort μ
endif
Update CMA parameters (ref alg 5)

Algorithm 4: Incorporation algorithm.

Update **C** matrix:

$s' = (1-c)s + c_u BD\mathbf{z}_{new}$

$\mathbf{C}' = (1-c_{cov})\mathbf{C} + c_{cov}\mathbf{s}\mathbf{s}^T$

Update σ:

$s'_\sigma = (1-c)s_\sigma + c_u BD\mathbf{z}_{new}$

$$\sigma' = e^{\left(\frac{\|s'_\sigma\|_2 - \hat{x}_N}{D\hat{x}_N} \right)}$$

Recalculate **B** and **D**:

B = Unit eigenvectors of **C** in columns.

D = Diagonal matrix of corresponding eigenvalues

Algorithm 5: CMA algorithm.

The tournament selection operator works according to algorithm 6, with the relationship operator (Pareto dominance calculation) given in algorithm 7. Note that the relationship operator includes an additional random artificial dominance in the case where two fitness vectors are exactly identical. While this case should almost never occur in practice, it has been necessary to include it when the evolutionary algorithm is used on some CFD cases involving large round off in the returned fitness values. The implementation is assisted by some pre-existing subroutines in Reference 31. The straightforward solution was found to simply be the random determination of dominance in the favour of one vector or another by coin toss.

```
Create an empty tournament population Q
for   i=1    to tournament size q  ,
            Choose at random from the selection buffer:
                        j=dice(B)
             Insert into the tournament population:
                        Q_i =B_j
loop
for   i=1…Q,,
    if  Relationship(I_new,Q_i ) < 0 : return false
loop
return true
```

Algorithm 6: Tournament selection.

```
Obtain two fitness vectors f and g in M objectives
if   f ≡ g ,  (random artificial dominance)
            if   coin(0.5)= heads return + 1 else return -1
endif
dom=0
if  f_1 < g_1,   dom= +1 else dom = -1
for   i=2, … M ,
             if    dom>0   and  g_i < f_i: dom = 0 and exit for-loop
             if    dom<0   and  f_i < g_i: dom = 0 and exit for-loop
             if    dom=0,
                         if  f_i < g_i: dom= +1
                         if  g_i < f_i : dom = -1
             endif
 loop
return dom
```

Algorithm 7: Relationship operator.

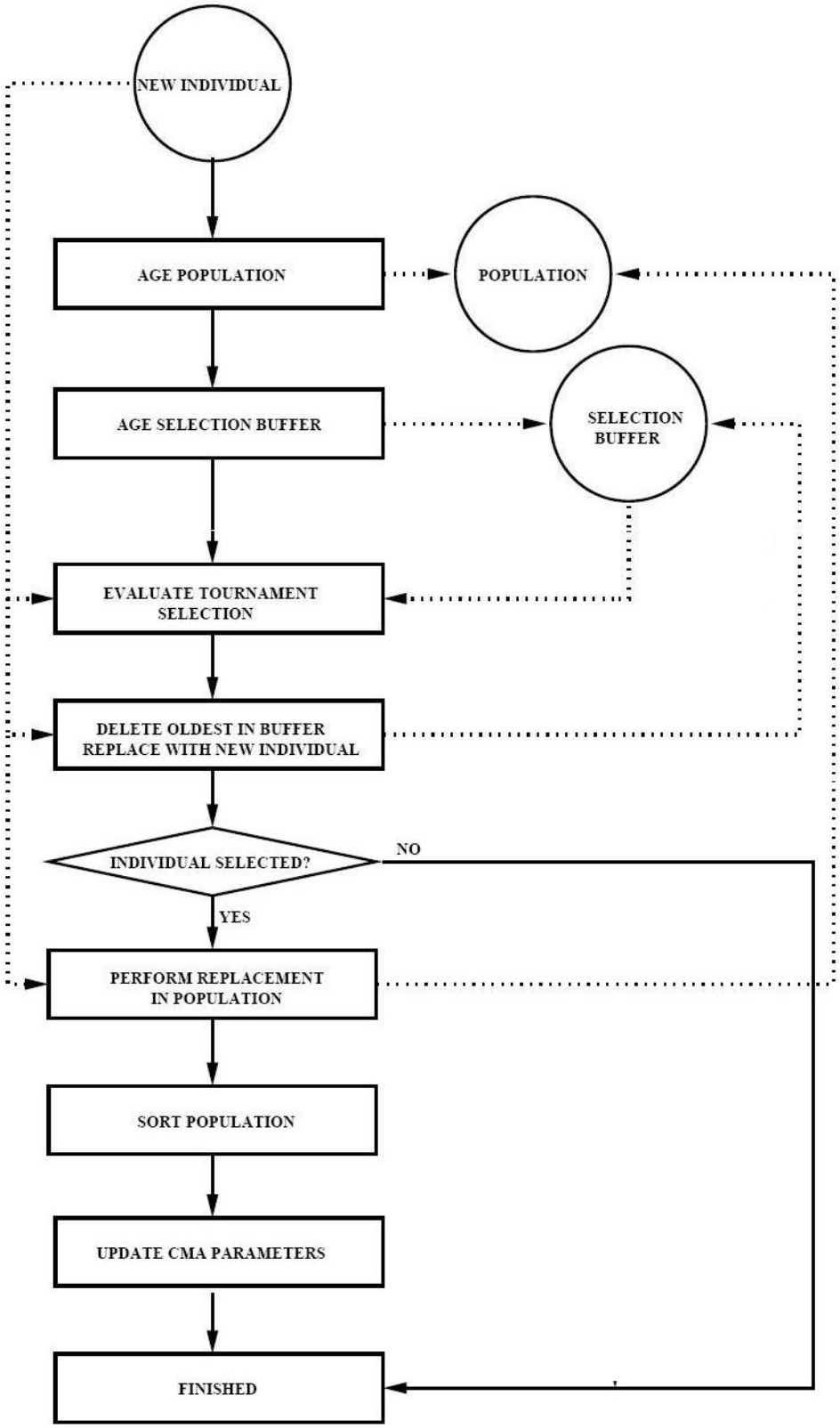


Figure 10: Overview of the incorporation process.

7.3. Generation of Individuals

When a new individual can be evaluated, it is generated as described in algorithm 8. This routine is the embodiment of the variation operator of the EA and a general overview is shown in figure 11. The generation routine notifies the host process (returns false) in the event that the individual exceeds any defined bounds, so that it may be rejected. The recombination operator (previous paper) returns both a recombined individual I_{new} (with object variable vector $\mathbf{x}_{new}$) and a measure of the distance between its parents ($dist(\rho)$). In this algorithm N is the number of problem unknowns, t is the current number of evaluations, T is the maximum number of evaluations allowed for the optimisation process and U and L are the upper and lower problem bounds.

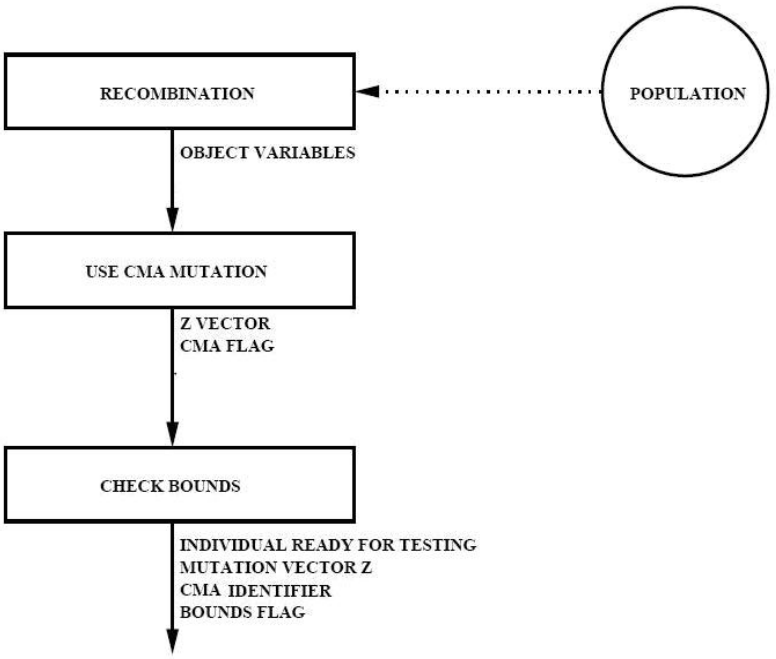
7.4 Stopping Condition

The stopping condition of any given problem depends strongly the nature of that problem. In direct optimisation problems such as shape design, the optimum fitness is generally not known *a priori*, except in very simple test cases. Therefore, the user must dictate when to halt a given run based on experience, and some knowledge of the convergence properties of the given algorithm. This is often specified as an upper limit on the number of function evaluations allowable, which in other words is essentially a time or cost limitation to the optimisation. In inverse optimisation situations, such as reconstruction or parameter fitting problems, the optimum fitness is generally known. This is due to the fact that the fitness is generally represented as an error, the difference between the candidate solution and the 'correct' solution, and often has an optimum value of zero.

In this case the stopping condition is generally specified as an allowable error bound on the fitness value, and when this fitness (or better) is reached, the run is stopped. Occasionally in these situations, a second stopping condition comprised of an upper limit on function evaluations is also imposed, so that in case a given fitness is never reached (due to some limitation in the optimiser or problem modelling procedure) the process does not run forever. In both the direct and inverse cases, the stopping condition must be specified by the operator, and therefore forms part of the stated problem as a whole.

```
x_new , dist(ρ))=recombine(μ)
Mutate offspring via CMA:
for   i=1,...N ,
          z_i = N(0,1)
loop
x'= x +BDz
Store z in I_new
endif
for i=1,...N,,
          if x'_i > U_i or  x'_i > L_i return false
loop
return true
```

Algorithm 8: Generation routine.



F**igure 11:** Overview of the generation process.

7.5 Integrated Multi-objective -Multidisciplinary Optimisation Formulation using Hierarchical Asynchronous Evolutionary Algorithms

When integrated with a hierarchical evolutionary algorithm, this analysis takes the form as illustrated in algorithm 9.This algorithm uses a hierarchical approach with three levels, on the bottom level a coarse type analysis to direct the exploration, at the top level more precise model that better describes the physics involved and at an intermediate level, a compromised balance between top and bottom layers is used. Initially the system will specify the design variables $\boldsymbol{x}$, constraints g_i, g_{ij} ,and parameters p , then it will generate random sub population of individuals μ_o at each layer, then it defines the number of subpopulations (nodes) i and number of hierarchical levels which for simplicity is equal to the number of analysis k. Once these initial populations are generated the algorithm will go through the steps described in the previous algorithms. Following the description in section 7.1, the scheduler first determines whether given stopping conditions (see section 7.4) have been met, and if so the evolutionary loop is exited and the entire process is stopped. If no stopping conditions are met, the scheduler updates the asynchronous solver (see section 6.2.1) so that further progress may be made. Then the scheduler determines whether or not candidate solutions which have been solved are ready for *incorporation* into the population. If such solutions exist, the *incorporation* routine (see section 7.2) is called and available candidates which now have had a fitness assigned are processed; it receives the individual, ages the population and buffer, performs Pareto tournament selection, deletes the oldest from the buffer and if the acceptance is true it is inserted in the population which I subsequently sorted, it then updates the CMA parameters (algorithm 5). Finally, the scheduler determines whether it is possible to generate more candidates, by polling the asynchronous solver. If this is possible, then the generation routine (see section 7.3) is called and individuals are generated via the evolutionary operators, by doing recombination, mutation via CMA, checking upper and lower bounds. During evaluation, the optimiser will take output analysis a_i and parameters p to guarantee satisfaction of constraints and compute the overall fitness function. If the problem is multi objective the algorithm will find the non-dominated individuals and will calculate the Pareto fronts.

Define design variables x parameters p , and constraints $g : Define(x, p, g_i, g_{ij})$

Define number of subpopulations (nodes) i , hierarchical levels and integrated analysis $k : Define(i,k)$

for all levels initialize subpopulations $(\mu_1^0, \mu_2^0, \mu_3^0, ..., \mu_n^0)$: $Analysis_k$

Layer 1: Uses Type 1 integrated analysis: $init(\mu_1^0)$: $Analysis_1$

Layer 2: Uses Type 2 integrated analysis: $init(\mu_2^0, \mu_3^0)$: $Analysis_2$

Layer 3: Uses Type 3 integrated analysis: $init(\mu_3^0, \mu_4^0, \mu_5^0, \mu_6^0)$: $Analysis_3$

loop

w*hile* stopping condition not met,

Update the asynchronous solver.

if solved individuals are available: incorporate them.

A new individual is received: I_{new}

Age population: For each μ_i in μ; $\mu'_{i\_age} = \mu_{i\_age} + 1$

Age selection buffer: For each b_i in B; $b'_{i\_age} = b_{i\_age} + 1$

Perform tournament selection: $accept = tournament\ (I_{new}, b)$

Delete oldest b_i in selection buffer, insert I_{new}

if $accept = true$

Perform replacement in μ: $replace\ (\mu, I_{new})$

Sort μ

endif

Update CMA parameters (ref alg 5)

if new individuals can be provided: generate them.

$\mathbf{x}_{new}$, $dist(\rho)) = recombine(\mu)$

Mutate offspring via CMA:

for $i = 1, ...N$,

$z_i = N(0,1)$

loop

$\mathbf{x}' = \mathbf{x} + BD\mathbf{z}$

Store $\mathbf{z}$ in $\boldsymbol{I}_{new}$

endif

for $i = 1, ...N,$,

if $x'_i > U_i$ or $x'_i > L_i$ *return* false

loop

return true

Evaluate candidate using specific integrated analysis type: $\lambda^{g+1} = f(\lambda_M^{g+1})$: $Analysis_i$

Get output analysis a_i , parameters p , check constraints g and add

…penalty: $\lambda^{g+1} = f(\lambda_M^{g+1}) + penalty$

if Multi-objective: Calculate Pareto fronts: Pareto $Pareto\ Front = Pareto(\lambda_M^{g+1})$

if epoch completed:

Start migration: $\mu_i^{g_k} = mig(\mu_i^{g_k} \rightarrow \mu_{i\pm1}^{g_k})$: $Analysis_i$

Layer 1: Receive best solutions from layer 2 reevaluate using Type 1 integrated analysis: $(\mu_1^{g_k}, \mu_2^{g_k} \rightarrow \mu_0^{g_k}), (\mu_o^{g_k} = f(\mu_o^{g_k}))$: $Analysis_1$

Layer 2: Receive random solutions from layer 1 and best from layer 3 reevaluate them using type 2 integrated analysis: $(\mu_0^{g_k} \rightarrow \mu_{1,2}^{g_k}, \mu_{3,45,6}^{g_k} \rightarrow \mu_{1,2}^{g_k}), (\mu_{1,2}^{g_k} = f(\mu_{1,2}^{g_k}))$: $Analysis_2$

Layer 3: Receive random solutions from layer 2 reevaluates them using type 3 integrated analysis $(\mu_{1,2}^{g_k} \rightarrow \mu_{3,45,6}^{g_k}), (\mu_{3,4,5,6}^{g_k} = f(\mu_{1,2}^{g_k}))$: $Analysis_3$

loop

Algorithm 9: Integrated MO-MDO formulation using EAs.

On a hierarchical topology with three levels, when the epoch is finished or the migration criteria is satisfied, the migration phase occurs: Layer 1 gets best solutions from Layer 2 and re-evaluates them using type of analysis one, Layer 2 gets random solutions from Layer 1 gets best solutions from Layer 3 and re-evaluates them using type of analysis two, Layer 3 gets random solutions from Layer 2 and re-evaluates them using type of analysis three. This process continues until a stopping condition is reached. These can be equal to a limited number of function evaluations, hours or a prescribed value on the fitness function.

8. IMPLEMENTATION OF MODEL TEST CASES

Methodology

In the sequel we will follow a design optimisation methodology that can be summarised with the following list of items:

1) *Problem Formulation*
2) *Definition of the EA Strategy: A Simple EA Single/Multi-objective EA, Parallel EA*
3) *Definition and Encoding of Design Variables*
4) *Definition of Constraints*
5) *Definition of Fitness Function*
6) *Definition of Analysis Tools-Software*
7) *Implementation –Design and Optimisation Rationale and Evaluation with EA Method*
8) *Optimisation Results –Pareto Fronts, Evolution Progress*
9) *Post-Processing of Final Solutions –Cp Distribution /Aero, Structural Performance.*

Problem Formulation

The first step problem formulation; when considering the solution and optimisation of a real-world engineering problem, the scientist/engineer usually tries to create a well-posed mathematical formulation which is representative of the problem at hand. The problem is that there are several complexities involved in the process; therefore some assumptions have to be made. This can be illustrated this with a simple example: the shape of the design and optimisation of an aircraft wing or a wing section; during the entire aircraft mission the wing is subjected to numerous flight conditions which are characterised by different Mach numbers, Reynolds numbers, angles of attack and other flight parameters. If we want to obtain the optimum shape for this wing, the design has to have good characteristics with regard to the aircraft payload, total mass, aerodynamic and structural performance. This is a complex task, therefore a sound solution to the problem is to make some assumptions and identify the main flight conditions, design parameters, constraints and then construct an optimisation problem in order to improve the performance.

In general, a shape optimisation problem can be formulated as a constrained minimisation problem of a cost function that involves the evaluation of a series of partial differential equations on a parameter-dependent geometrical domain. The basic problem consists in finding the optimal shape in a search space of feasible shapes while satisfying non-linear design constraints. This optimum shape then has to minimise an objective or multiple objectives that depend on the selected shape and that conform to the solution of the partial differential equations corresponding to the disciplinary or multidisciplinary analysis. In general, the optimisation problem can be defined as:

$$P = \left\{ \begin{array}{c} \text{Find a shape } \Omega^* \text{ such that} \\ \mathrm{J}(\Omega^*) = \min\, j(\Omega) = \min\, j(\Omega, W_\Omega) \\ \text{under the constrants } S(\Omega) \le \varepsilon_o \\ \text{and with the state } W_\Omega \text{ solution of the equation} \\ j(\Omega, W_\Omega) = 0 \end{array} \right\} \quad (5)$$

where:

- Ω denotes the unknown shape and is a subset of $\Re^3$ is defined in general by a finite set of parameters called the design variables,
- W_Ω are the state variables that characterise the state of the system under study. In general, W_Ω is the unique solution of a set of partial differential equations.
- $j(\Omega) = j(\Omega, W_\Omega)$ is the objective or cost function which must be minimised.
- $S(\Omega)$ defines the set of constraints imposed on the shape.

Definition of the EA Strategy: A Simple EA Single/Multi-objective EA, Parallel EA.

A second step is the selection of an optimisation strategy. As discussed in section 1, even though traditional gradient based methods are fast but the complexity, non-linearity and multi-objective characteristics of some problems ay require the use of more sophisticated optimisers. Also, the computational cost of a Navier-Stokes or Euler solution around a high lift device, 3D wing for example involves high computational expense therefore it is also desirable to use parallel computations and a multi-fidelity approach. In these cases we want to use an efficient and robust parallel multi-objective EA (PMOEA).

Definition and Encoding of Design Variables

The selection of an appropriate definition of design variables and geometric representation that accounts for the complexities of the design space is the third step. In aerofoil shape optimisation for example we could use Bézier or Spline curves or a PARSEC representation the blade contour. In the Bézier curves representation, for the design variables, we use a combination of mean line and thickness distribution control points (see section 8.3.1 for more details). Similar to the NACA series aerofoils, the PARSEC [36] parameterises upper and lower surface of the aerofoil using polynomial in coordinates *X, Z* as:

$$Z = \sum_{n=1}^{6} a_n . X^{n-1/2} \quad (6)$$

Where a_n are real coefficients. The PARSEC aerofoils are defined by basic geometric parameters: leading edge radius (r_{le}), upper and lower crest locations including curvatures (X_{us}, Z_{us}, Z_{xxus}, X_{ls}, Z_{ls}, Z_{xxls}) trailing edge ordinate (Z_{te}), thickness (ΔZ_{te}) and directions and wedge angles (α_{te}, β_{te}). These parameters can be expressed by the original coefficients a_n by solving a simple set of simultaneous equations. Additional details can be found in Reference [36].

Definition of Constraints

In this step the user needs to specify a series of constraints, such as maximum and minimum thickness location, pitching moment and lower and upper bounds for the Bézier/Spline control points.

Definition of Fitness Function

The fitness function is specified by the end user of the optimiser. It is referred to by many other names, such as the cost function and the payoff function. It may be purely mathematical, the result of some solution process (Computational Fluid Dynamics, Finite Element Analysis, etc) or be the result of an experiment. Formally, the fitness function is simply defined:

$$\mathbf{f} = f(\mathbf{x}) \tag{7}$$

Where **x** is the vector of object variables and **f** is the resulting vector of fitness values. Neither **f** nor ***x*** can be empty vectors. One vector of object variables is associated with each individual, so it is also legal to write: $\mathbf{f} = f(I)$.
Further, it can be applied to each member of a population, so we may also write: $\mu_{eval} = f(\mu)$. The distinction between problems in single and multiple objectives occurs here, with the fitness vector only having one element for a single objective problem.

Definition of Analysis Tools-Software

The next step is to decide which type and fidelity of solver to use. A decision has to be made in order to compromise on the use of a higher fidelity solver, such as a Navier-Stokes solver/Finite Element Analysis which are computationally expensive but accurate, the use of lower fidelity models, such as Euler or panel or analytical methods, which are fast but could be unstable, or a combination of both. The structural properties can be computed with a combination of FEA and an analytical method. Another alternative is on using a single model but defining different grid densities.

Implementation –Design and Optimisation Rationale

In this step it is recommended to develop an optimisation algorithm. An example of an optimisation rationale is illustrated in algorithm 10. Initially the design variables, design constraints, flow conditions and fitness functions are defined and then an initial population of wing geometries is generated at random. Then while the termination criteria has not been satisfied, the algorithm generates shapes, evaluates, recombines and mutates individuals, computes Pareto fronts and proceeds with migration process. The termination criterion is satisfied when the prescribed number of function evaluations is reached or when the fitness value goes below a prescribed number. The last stage is the designation of result such as best-so-far individual, non-dominated individual so called Pareto-front and then the optimisation process will be terminated.

Optimisation results –Pareto fronts, Evolution Progress

This refers to the importance of keeping track of the statistics of the evolution process, number of generations, Pareto fronts but without adding major computational burden to the optimisation process. These are for example plots for value of fitness function versus evolution process or a plot of the current non-dominated solutions and whole population. If we are comparing the Pareto solutions and Nash solutions, a plot of this is also desirable.

Post-processing of Final Solutions –Cp Distribution /Aero, Structural Performance.

This is related to what is interesting to the designer; these can be for example comparing shapes in the Pareto optimal front, comparing the *Cp* distribution over an aerofoil, and/or comparison of aerodynamic performance of baseline geometry and optimal solutions.

Initialise Populations at each level of the hierarchical tree: $init(\mu^0)$
Evaluate using analysis tool at this level: $f(\mu^0)$
$g=0$
while stopping condition not met,
  Recombine: $\lambda_R^{g+1} = reco(\mu^g)$
  Mutate: $\lambda_M^{g+1} = mut(\lambda_R^{g+1})$
  Evaluate: Evaluate using analysis tool at this level: $\lambda^{g+1} = f(\lambda_M^{g+1})$
  Selection: Select individuals using tournament selection
  Pareto Fronts: Compute Pareto fronts
  Migration check: If migration criteria satisfied, migrate up best solutions at each level, migrate down random solutions.
$g=g+1$
loop
Post-Processing: Post-process optimisation results and optimal high lift configurations, (i.e. Pareto fronts graphs, evolution progress, *Cp* distribution, mach contour, pressure contours for selected members of the Pareto front

Algorithm 10: General Algorithm.

8.2 Mathematical Test Cases

In this section we apply the concepts and steps in the previous section to test the performance of the method for mathematical test cases to determine expected performance levels of the algorithm in real situations. The test cases are all posed as easily closed-form mathematical functions. There are a great number of test cases for single and multiple objective results presented in the literature, and are almost as varied as the number of evolutionary algorithms that have been written. Most algorithms used today have been written with a single objective in mind, and the vast majority of early evolutionary algorithms were only written with a single objective capability. It is of course impossible to be completely general when establishing a test suite, so some subjective viewing of the matter is required, and we must make some generalising assumptions. This section address problems with two objectives, but application of the method to single objective problems can be found elsewhere [14, 41, and 43].

We will consider the convergence of the algorithm to a known Pareto front. It is difficult to directly measure the performance of an algorithm in completing a multi-objective task as the metrics to define multi-objective performance are not clear. Some work using closed-form expressions has been performed by Deb [9]. It is evident when the entire population has converged to the front; this can be seen by inspection. Because inevitably we consider a discrete approximation to a smooth surface (in the case of mathematical test functions), opinions vary regarding the fidelity of approximation that is achieved. The maximum number of points available to correctly model the Pareto front in this work is equal to the population size. It is desirable to have a god spread of points along the Pareto front; however this can be defined both in objective variable space and in fitness space. Because there is an explicit distortion caused by mapping between the two spaces, a method that provides an even distribution in fitness space will not provide this in objective variable space, and vice--versa.

In this work the authors have adopted the fitness space approach, due to the belief that the design engineer invariably would like to obtain a distribution of 'figures of merit' in advance, rather than a distribution of actual geometries. In any case, it is quite difficult to subsequently define a 'density distribution' of solutions along the Pareto front with a hope for providing an even

positioning of solutions. The study on multi-objective test functions is based on works by Deb [9] and Coello-Coello *et al.* [7]. In these references a comprehensive set of test functions and a study of a set of problems highlight the difficulties of a multi-objective algorithm to converge. For illustration purposes we consider only one test case but additional test cases can be found in References 14-17, 41 and 43.

TNK

Problem Formulation
The TNK is an example of a constrained problem that has five discontinuous Pareto optimal fronts. The problem has two variables and two constraints.

Definition of the EA Strategy: A Simple EA Single/Multi-objective EA, Parallel EA.
In this case a conventional deterministic optimiser will encounter difficulties or will fail. We need a robust optimiser. In this case we select the HAPEA algorithm with a single population but NSGA or VEGA work well [7, 9].

Design and Encoding of Design Variables
The problem has two design variables. The upper and lower bounds are:

$$\begin{aligned} 0 \le x_1 \le \pi, \\ 0 \le x_2 \le \pi \end{aligned} \tag{8}$$

Similar to previous case we define an individual as an object containing a vector of the 30 problem variables (**x**), a vector of strategy variances (σ) and rotation angles (α):

$$I=(\mathbf{x},\ \sigma,\ \alpha)=((x_1,x_2),(\ \sigma_1,\sigma_2,\ldots,\ \sigma_N),(\ \alpha_1,\alpha_2,\ldots.,\ \alpha_{N(N-1)/2})) \tag{9}$$

Fitness Functions
Following Deb [9] we define the fitness functions f_1 and f_2 as follows:

$$f_1(x) = x_1 \tag{10}$$

$$f_2(x) = x_2 \tag{11}$$

Constraints
The problem has two constraints:

$$C_1(x) \equiv x_1^2 + x_2^2 - 1 - 0.1\cos\left(16\arctan\frac{x_1}{x_2}\right) \ge 0 \tag{12}$$

$$C_2(x) \equiv (x_1 - 0.5)^2 + (x_2 - 0.5)x_1^2 \le 0.5 \tag{13}$$

Analysis Tools:
In this case the analysis tool or solver is simply an analytical expression described in expressions (10) – (13).

Implementation –Design and Optimisation Rationale
The problem was implemented using the HAPEA approach with one single level, a population size of 100 and discrete recombination; we take care of constraints by a penalty criteria and use the following optimisation algorithm:

Initialise Populations: $init(\mu^0)$
Evaluate each individual of the initial population $f(\mu^0)$
$g=0$
while stopping condition not met,
  Recombine: $\lambda_R^{g+1} = reco(\mu^g)$
  Mutate: $\lambda_M^{g+1} = mut(\lambda_R^{g+1})$
  Evaluate: $\lambda^{g+1} = f(\lambda_M^{g+1})$
  Selection: Select individuals using tournament selection
  Pareto Fronts: Compute Pareto fronts
$g=g+1$
loop
Post-Processing: Post-process optimisation results, comparison of True and Computed Pareto Front.

Algorithm 11: TNK Algorithm.

Optimisation Results

Figure 12 shows a comparison of actual and computed Pareto optimal fronts. It is shown that the algorithm has correctly distributed all individuals across the Pareto front. The solutions are very evenly spread across the front.

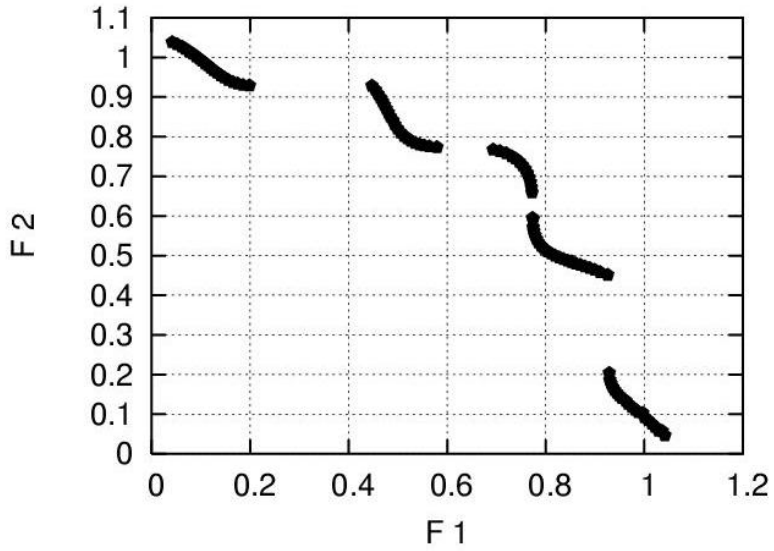


Figure 12: Pareto Front TNK

8.3 Conceptual Design: Aeronautical Design Test Cases

In this section, some results on the application of the method for conceptual design are presented to find either correct or compromised solutions to multi-objective problems. Other examples can be found in Part 3 and References 14, 15 and 16.

In conceptual design, basic questions such as size, weight, configuration arrangement and performance are answered with simplified software and rules of the thumb assumptions making this approach cheap and flexible to the designer. In this phase, a large number of possible alternatives are evaluated together with trade studies. In conceptual design, the design requirements are evaluated and studied to guide and evaluate different aircraft configurations. In conceptual design we might also know the target pressure distribution over a wing or an aerofoil and want to validate or test the optimisation algorithm. Conceptual design is a very fluid process

and the configuration layout is changed constantly to incorporate new concepts and re-evaluate potential improvements. Trade studies and sophistication of design are improved as the concept design progresses in time. As the level of complexity increases, the level of detail increases and new phases of the design are considered.

8.3.1 Aerofoil Reconstruction: Two Aerofoils at Two Different Design Points

Practising engineers in aerodynamics design often know an appropriate pressure distribution for their problems, but do not always have tools for finding a feasible solution. This problem has been studied extensively by Jameson [20,21], Kim and Rho [22], and Obayashi [27], Obayashi and Takanashi [28] and Oyama *et al*. [29] amongst many others.

Problem Formulation
This case considers minimisation of the difference on the surface pressure coefficient distribution on a candidate aerofoil and the pressure distribution over two aerofoils (NACA0012 and RAE2822) operating at different design points. The flow conditions for the two points analysed are:

Flow Condition 1: *$M_\infty = 0.2$, $Re = 2.7 \times 10^6$, Angle of Attack = 1.25 (NACA0012)*
Flow Condition 2: *$M_\infty = 0.75$, $Re = 9.0 \times 10^6$, Angle of Attack = 1.25 (RAE2822)*

Definition of the EA Strategy: A Simple EA, single objective EA, Multi-objective EA, Parallel EA.
This problem would have probably been solved faster using a conventional deterministic optimiser if the two objectives where sought independently. In this case we want to find the Pareto in one single run and without having to determine wights in advance.

Definition and Encoding of Design Variables
The aerofoil geometry is represented by two Bezier curves, one for the mean line and one for the thickness distribution. The mean line--thickness distribution is a standard method for representing aerofoils [1], as it closely couples the representation with the results; the mean line has a powerful effect on cruise lift coefficient and pitching moment, while the thickness distribution has a powerful effect on the cruise drag. Put simply, the aerofoil is obtained by perpendicular offset of the thickness distribution about the mean line.

For a given mean line point (x_m , y_m) and matching thickness distribution height y_t, an upper and lower surface point can be obtained:

$$x_{u,l} = x_m \pm y_t \sin(\theta) \tag{14}$$

$$y_{u,l} = y_m \pm y_t \cos(\theta) \tag{15}$$

where θ is the angle of the mean line at (x_m,y_m). This is shown in figure 13. We select the x-positions of the Bezier control points in advance; the y- positions remain as the unknowns. The only restrictions are that the first and last points are fixed to (0, 0) and (1, 0) to provide leading and trailing edges respectively, and that the first control point on the thickness distribution must be directly above the leading edge (i.e. $(0,y_{c,1})$) to provide a rounded geometry (Bezier curves are by definition always tangent to the extreme edges of their defining envelopes).
We bound the vertical heights to range $y_c \in [0.01,0.10]$ giving a very wide range of possible geometries (theoretically spanning aerofoils from 2% to 20% thick).The advantage of using single high--order Bezier curves for the representation rather than piecewise splines or others is their geometric stability. A Bezier curve must by definition always be contained within the bounding envelope of control points. Furthermore, if the bounding envelope is not re--entrant, then the curve will also have this property. Also, Bezier curves do not 'kink' like a piecewise spline, and

the defining equations are not stiff (ill-conditioned). Therefore, a small change in control point location will always result in a small change in surface representation. This provides a favourable interface between the optimiser and the flow solver. For this case, four evenly spaced interior (free) control points were taken for the mean line, and five for the thickness distribution, giving a problem in nine unknowns. We define an individual as an object containing a vector of the 9 problem variables (**x**), a vector of strategy variances (σ) and rotation angles (α):

$$I=(\mathbf{x},\ \sigma,\ \alpha)=((y_1, y_2, y_3, y_4, y_1', y_2', y_3', y_4', y_5'),(\ \sigma_1, \sigma_2, \ldots,\ \sigma_N),(\ \alpha_1, \alpha_2, \ldots.,\ \alpha_{N(N-1)/2}))$$

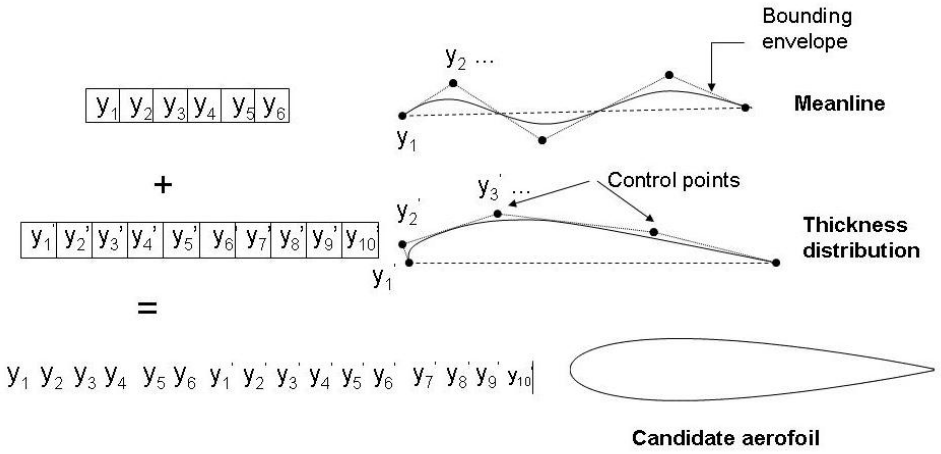


Figure 13: Aerofoil Representation using two Bezier curves

Fitness Functions

The fitness functions are the Root Mean Square Error (RMSE) of the surface pressure coefficients distribution between candidate (current) and target –one (objective one) and surface pressure coefficients between candidate (current) and target –two (objective two). The problem is solved when the positive value of the fitness goes below a prescribed value.

$$\min(f_1): f_1 = \frac{1}{N}\sqrt{\sum_{i=1}^{N}\left(Cp_{current} - Cp_{Target-one}\right)^2} \tag{16}$$

$$\min(f_2): f_2 = \frac{1}{N}\sqrt{\sum_{i=1}^{N}\left(Cp_{current} - Cp_{Target-two}\right)^2} \tag{17}$$

Analysis Tools

In this case we need a solver to compute the aerodynamic drag. We select the MSES solver, [11] which is based on a structured quadrilateral streamline mesh which is coupled to an integral boundary layer based on a multi-layer velocity profile representation.

Implementation –Design and Optimisation Rationale

The problem was implemented using the procedure described in algorithm 12:

Details on the multi-fidelity –hierarchical tree are: (EA with CMA/Pareto tournament selection, Asynchronous Evaluation) on each node of the hierarchical tree with the following parameter settings for the EA and CFD solver are:

Top Layer: A population size of 20, intermediate recombination used between two parents, and a mesh of *215 x 36*.

Middle Layer: A population size of 20, discrete recombination used between two parents, and a mesh of *165 x 27*.

Bottom Layer: A population size of 20, discrete recombination used between two parents, and a maximum of *151 x 24*.

Optimisation Results and Post-processing of Optimal Solutions

This problem was run for 3000 function evaluations of the head node, and took approximately ten hours on the cluster with twelve machines. Figure 14 shows the Pareto front obtained for this test case. Figure 15 shows a comparison of the target geometries and figures 16 and 17 surface pressure distributions respectively. As illustrated, there is a good match on the computed and target surface pressure distribution.

Initialise Populations at each level of the hierarchical tree: $init(\mu^0)$
Evaluate using analysis tool at this level- see below: $f(\mu^0)$
$g=0$
while stopping condition not met,
  Recombine: $\lambda_R^{g+1} = reco(\mu^g)$
  Mutate: $\lambda_M^{g+1} = mut(\lambda_R^{g+1})$
  Evaluate: Evaluate using analysis tool at this level: $\lambda^{g+1} = f(\lambda_M^{g+1})$
  Selection: Select individuals using tournament selection
  Pareto Fronts: Compute Pareto fronts
  Migration check: If migration criteria satisfied, migrate up best solutions at each level, migrate down random solutions.
  $g=g+1$
loop
Post-Processing: Post-process optimisation results and optimal aerofoil shapes (i.e. Pareto fronts graphs, evolution progress, *Cp* distribution for each aerofoil.

Algorithm 12: Design Optimisation Rationale for Aerofoil Reconstruction Problem.

Performance with Increasing Number of Computers

In this section we use the previous test case and consider the performance of the algorithm with an increasing number of computers. The parallel environment used is the BORGS cluster of PCs. The message-passing model used is the Parallel Virtual Machine (PVM) [12]. Figure 18 shows the speed-up of the computation as the number of computes increases and comparison with a linear speed-up for reference purposes. These test case, clearly show that the method benefit from the use of parallel computing strategies.

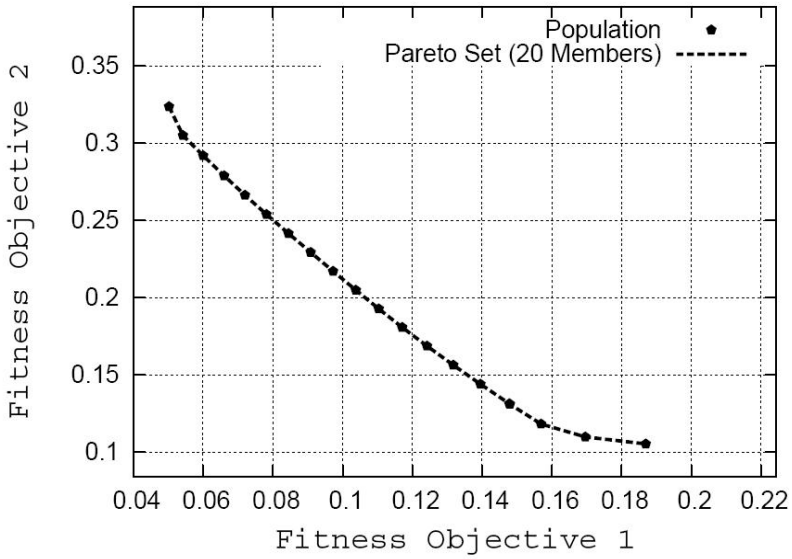


Figure 14: Pareto front for multi-point aerofoil design.

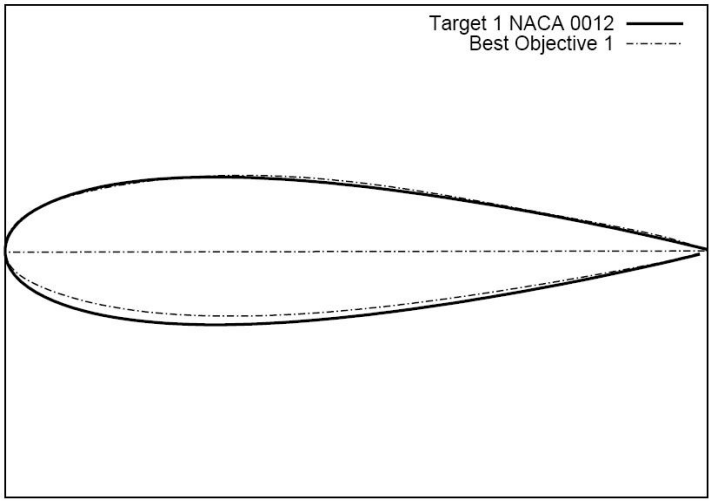


Figure 15: Target and computed geometries, multi-point aerofoil design, objective one.

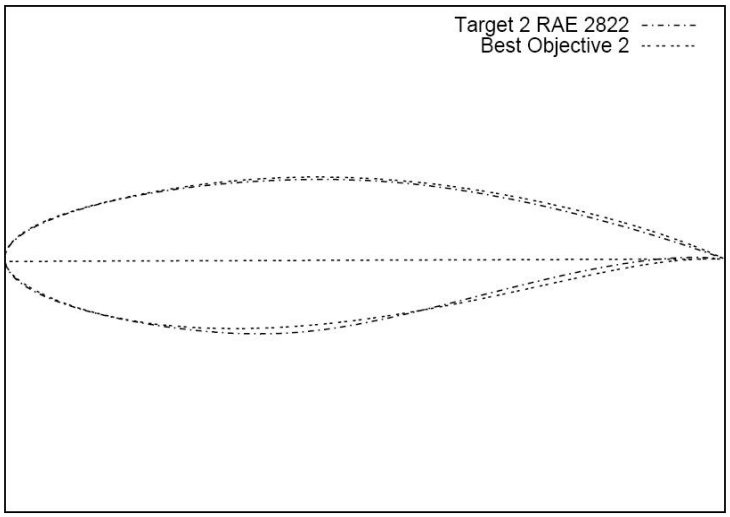


Figure 16: Target and computed geometries, multi-point aerofoil design, objective two.

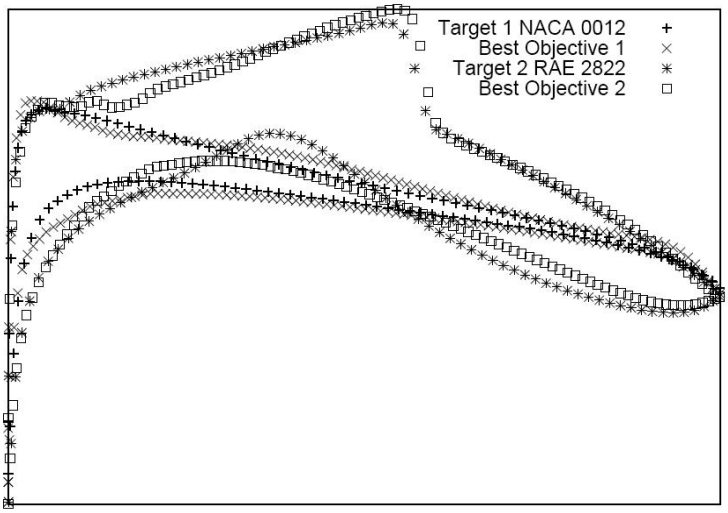


Figure 17: Target and computed pressure distribution for multi-point aerofoil design.

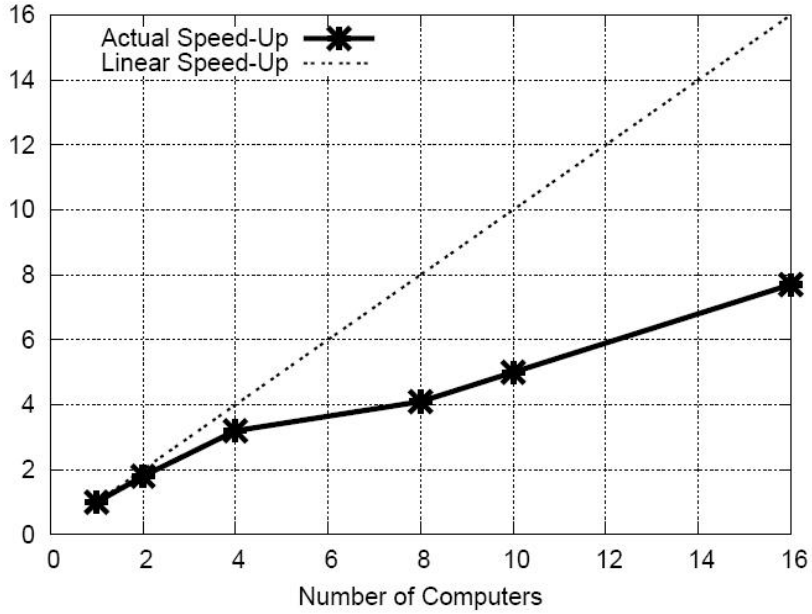


Figure 18: Speed-up of the computation with increasing number of computers and comparison with linear speed-up.

8.3.2 Test Case 2: Two Objective UAV Aerofoil Section Optimisation

This test case is explained on a step by step approach to attendees of the VKI lecture series on Introduction to Optimization and Multidisciplinary Design in Aeronautics and Turbomachinery, 31 May to 4 June 2010.

The use and development of Unmanned Aerial Vehicles for military and civilian applications are rapidly increasing but there are difficulties in the design of these vehicles because of the varied and non-intuitive nature of the configurations and missions that can be performed. Similar to their manned counterparts, the challenge is to develop trade-off studies of optimal configurations to produce a high performance aircraft that satisfy mission requirements.

Problem Formulation

In this second test, we consider the detailed design of a single element aerofoil for a small UAV application similar to the RQ-7A Shadow 200 Tactical UAV and use the aerofoil design module for this task. The aircraft maximum gross weight is approximately 320 *Lbs*, it has a wingspan of approximately 12.8 *ft*, a mean chord of approximately 2 *ft,* length of 11 *ft,* and a planform shape without sweep. We assume the aircraft operating between a slow cruise 33.3 *m/s* and fast cruise 46.6 *m/s* approximately. This result in the airframe, flight parameters and operating conditions indicated in table 1. These conditions assume an aircraft at mid weight-cruise during and extended cruise phase at intermediate altitude.

Aerofoil section	NACA4415
b, *ft*	12.8
$\boldsymbol{c_r}$ (aprox), *ft*	2.0
Length, *ft*	11.2
Cruising altitude, *m*	3000

Description	Flight Condition One — Slow Cruise	Flight Condition Two — Fast Cruise
M_∞	0.1025	0.141
Re	1.085×10^6	1.490×10^6

c_l	1.18	0.6140

Table 1: UAV Data and Operating Conditions.

For the optimisation, we initially assume an existing aerofoil geometry operating at two suggested design points, and then design and aerofoil that preserves the original thickness while reducing the drag coefficient. The assumed baseline aerofoil geometry is the *NACA4415*. This aerofoil is *15%* thick. Figures 19 and 20 show the pressure coefficient (*Cp*) distribution and some aerodynamic data for the two flight conditions considered. The combined polars for the NACA4415 aerofoil are shown in Figure 21. It is noted that both cruise points operate inside the invariant drag region of the aerofoil; the low speed cruise condition giving approximately c_d = *0.016* and the high speed giving approximately c_d =*0.012*.

Definition of the EA Strategy: A simple EA, single objective EA, Multi-objective EA, Parallel EA.

The complexity, non-linearity and multi-objective characteristics of this problem make it suitable to be solved by an EA optimiser. The computational cost for this problem is an important consideration as we would like open wide upper and lower bounds in the search space and depends of the computing facilities used, in particular in industrial design environments. Therefore it is also desirable to use parallel computations and a multi-fidelity approach. In this case we want to use a multi-objective parallel EA (MOPEA) and select the HAPEA approach which has all these capabilities but other EA approaches would perform well also like NSGAII or VEGA [9] for example.

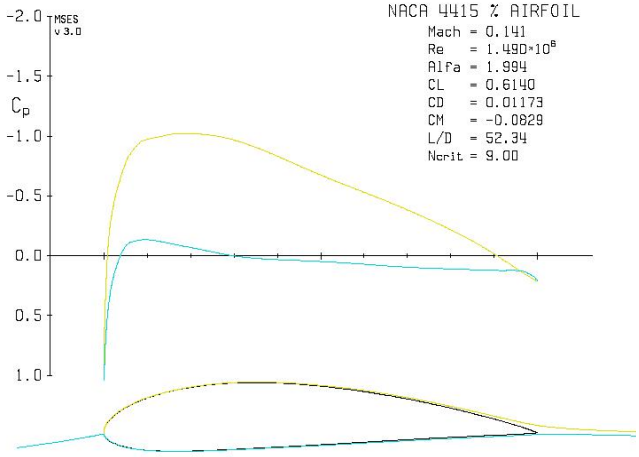


Figure 19: NACA 4415 - Flight Condition Two — Fast Cruise**.**

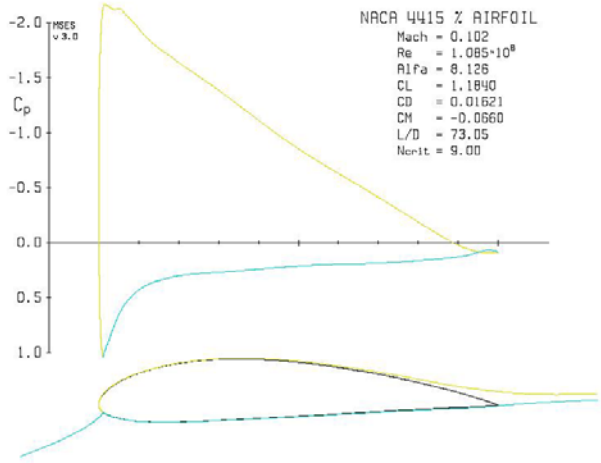


Figure 20: NACA 4415 - Flight Condition One — Slow Cruise.

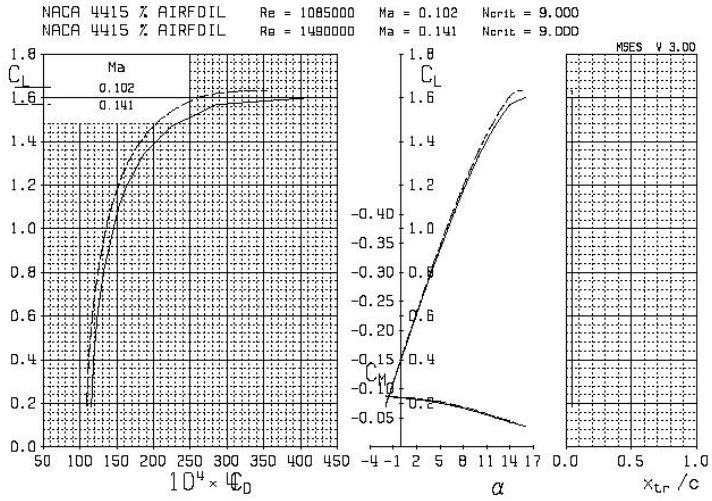


Figure 21: NACA 4415 – Polar

Design Variables

The aerofoil geometry is represented by two Bezier curves, one for the mean line and one for the thickness distribution. The mean line--thickness distribution is a standard method for representing aerofoils [1], as it closely couples the representation with the results; the mean line has a powerful effect on cruise lift coefficient and pitching moment, while the thickness distribution has a powerful effect on the cruise drag. In this case we consider six free control points on the mean line and ten free control points on the thickness distribution.

Encoding of Design Variables-Shape Parameterisation

We define an individual as an object containing a vector of the 16 problem variables, a vector of strategy variances (σ) and rotation angles (α) for the evolution strategy:

$$I=(\mathbf{x}, \sigma, \alpha) =$$
$$((y_1, y_2, .. , y_6, y_1', y_2' .... , y_{10}'),(\sigma_1, \sigma_2 , ... , \sigma_N),(\alpha_1, \alpha_2 , ..., \alpha_{N(N-1)/2}))$$

Fitness-Objective Functions

The two fitness functions to be optimised are defined as minimisation of drag (c_d) at the two flight conditions.

$$\min(f_1) = c_d \quad Re = 1.085 \times 10^6 , c_l = 1.184 \tag{3}$$

$$\min(f_2) = c_d \quad Re = 1.490 \times 10^6 , c_l = 0.6140 \tag{4}$$

Design Constraints

There are three types of constraints: maximum thickness, maximum thickness location and pitching moment (c_m). The thickness and maximum thickness location of each aerofoil must exceed 15 % ($t/c \geq 0.15$) and be between 20 and 40% chord, respectively. If a constraint on pitching moment is applied this must not be more severe than -0.0660 ($c_m \geq -0.0660$). When all constraints are considered they are added up and applied by equally penalising both fitness values via a linear penalty method.

Analysis Tools

The aerodynamic characteristics of the candidate aero foils are evaluated using the *MSES* [11] software. This solver is based on a structured quadrilateral streamline mesh which is coupled to an integral boundary layer based on a multi layer velocity profile representation. Details on *MSES* can be found in Drela [11]

Implementation - Design and Optimisation Rationale

In designing a replacement aerofoil for this UAV platform, the following design factors are considered:

- Maintain approximately the same c_l so as to not impinge upon the assisted launch and landing length.
- Maintain at least the current thickness, so as not to increase the weight of the wing.
- Lower the drag at both cruise points, in a multi-objective fashion.
- Study the implication of constraining the pitching moment coefficient during the evolutionary optimisation.

The problem was implemented using the following optimisation algorithm coupled with a Pareto game strategy:

Define design variables x parameters p, and constraints $g: Define(x, p, g_i, g_{ij})$

Define number of subpopulations (nodes) i, hierarchical levels and integrated analysis $k: Define(i,k)$

for all levels initialize subpopulations $(\mu_1^0, \mu_2^0, \mu_3^0, ..., \mu_n^0)$: $Analysis_k$

Layer 1: Uses Type 1 integrated analysis: $init(\mu_1^0)$: $Analysis_1$

Layer 2: Uses Type 2 integrated analysis: $init(\mu_2^0, \mu_3^0)$: $Analysis_2$

Layer 3: Uses Type 3 integrated analysis: $init(\mu_3^0, \mu_4^0, \mu_5^0, \mu_6^0)$: $Analysis_3$

loop

w*hile* stopping condition not met,

Update the asynchronous solver.

if solved individuals are available: incorporate them.

if new individuals can be provided: generate them.

Evaluate candidate using specific integrated analysis type: $\lambda^{g+1} = f(\lambda_M^{g+1})$: $Analysis_i$

Get output analysis a_i, parameters p, check constraints g and add

…penalty: $\lambda^{g+1} = f(\lambda_M^{g+1}) + penalty$

if Multi-objective: Calculate Pareto fronts: Pareto $Pareto\ Front = Pareto(\lambda_M^{g+1})$

if epoch completed:

Start migration: $\mu_i^{g_k} = mig(\mu_i^{g_k} \rightarrow \mu_{i\pm1}^{g_k})$: $Analysis_i$

Layer 1: Receive best solutions from layer 2 reevaluate using Type 1 integrated analysis: $(\mu_1^{g_k}, \mu_2^{g_k} \rightarrow \mu_0^{g_k}), (\mu_o^{g_k} = f(\mu_o^{g_k}))$: $Analysis_1$

Layer 2: Receive random solutions from layer 1 and best from layer 3 reevaluate them using type 2 integrated analysis: $(\mu_0^{g_k} \rightarrow \mu_{1,2}^{g_k}, \mu_{3,45,6}^{g_k} \rightarrow \mu_{1,2}^{g_k}), (\mu_{1,2}^{g_k} = f(\mu_{1,2}^{g_k}))$: $Analysis_2$

Layer 3: Receive random solutions from layer 2 reevaluates them using type 3 integrated analysis $(\mu_{1,2}^{g_k} \rightarrow \mu_{3,45,6}^{g_k}), (\mu_{3,4,5,6}^{g_k} = f(\mu_{1,2}^{g_k}))$: $Analysis_3$

loop

Post-Processing: Post-process optimisation results and optimal aerofoil shapes (i.e. Pareto fronts graphs, evolution progress, Cp distribution for each wing in the Pareto front, aerodynamic performance-CD,CM, CL- for selected wings from the Pareto front

Algorithm 2: Design Rationale for UAV Aerofoil Optimisation.

Details on the multi-fidelity –hierarchical tree are: (EA with CMA/Pareto tournament selection, Asynchronous Evaluation) on each node of the hierarchical tree with the following parameter settings for the EA and CFD solver:

We use the same parameter settings on the evolutionary optimisation algorithm for the two test cases considered:

Top Layer: A population size of 20 and a computational mesh of 215×36.

Middle Layer: A population size of 20 and a computational mesh of 165×27.

Optimisation Results and Post-processing of Optimal Solutions

This test case was run for 2000 function evaluations on the top level and took approximately 8 hours in a cluster of 4 2.8 GHZprocessors. Figure 22 and 23 shows the ensemble of aerofoils in the Pareto front. From this front, we select three aerofoils; objective one optimal, objective two optimal and compromise aerofoil from the middle of the front. These geometries are shown against the *NACA 4415* aerofoil in Figure 20.

We consider a multi-element aerofoil from the middle of the Pareto front for further evaluation. Figures 24 and 25 show the *Cp* distribution for the two flight conditions. Figure 26

shows the comparative drag polars for $Re=1.480 \times 10^6$ and Figure 27 that for $Re=1.085 \times 10^6$. But in this case a lower c_m is obtained for both conditions. Table 2 summarizes the drag reduction at the two flight conditions for the two test cases considered.

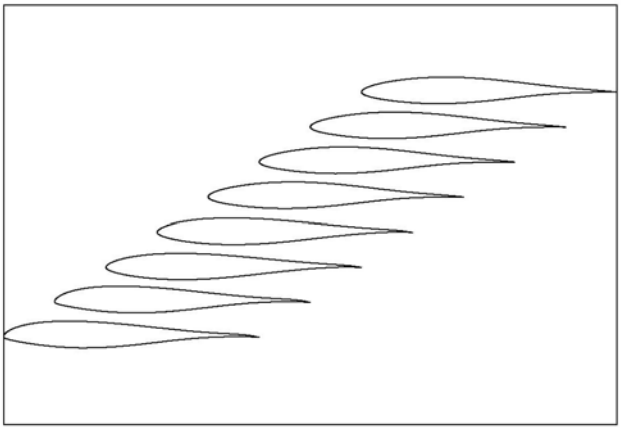


Figure 22: The Ensemble of Pareto Aerofoils - c_m Constrained.

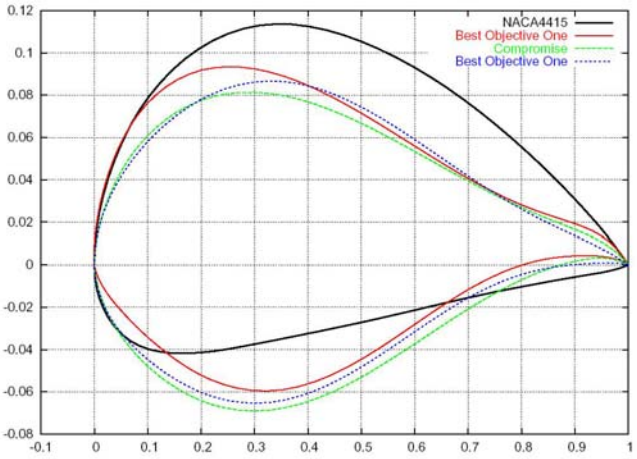


Figure 23: Comparison of selected geometries - c_m Constrained

.

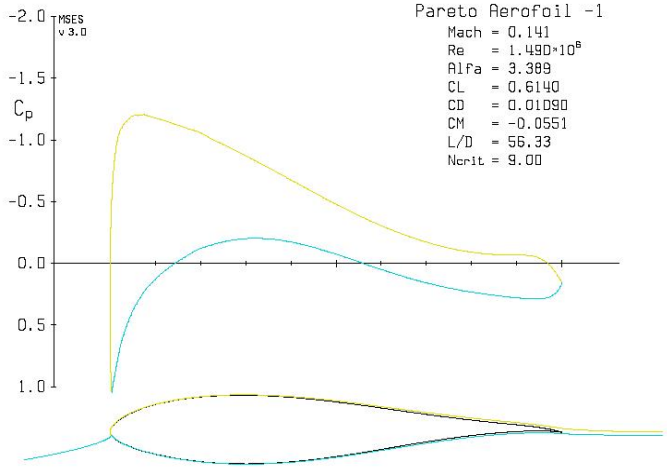


Figure 24: Pareto 01 Flight Condition One - c_m Constrained

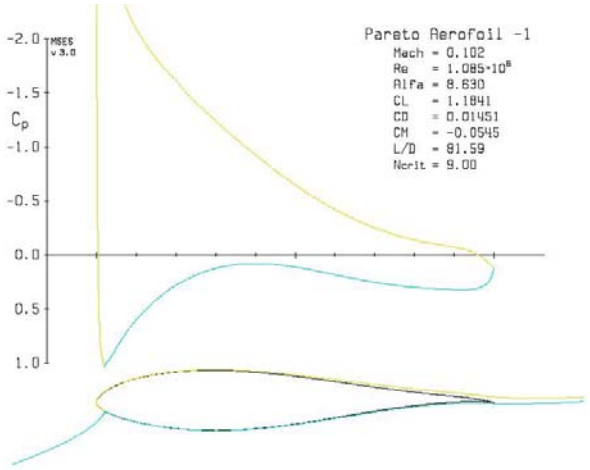


Figure 25: Pareto 01 Flight Condition Two - c_m Constrained

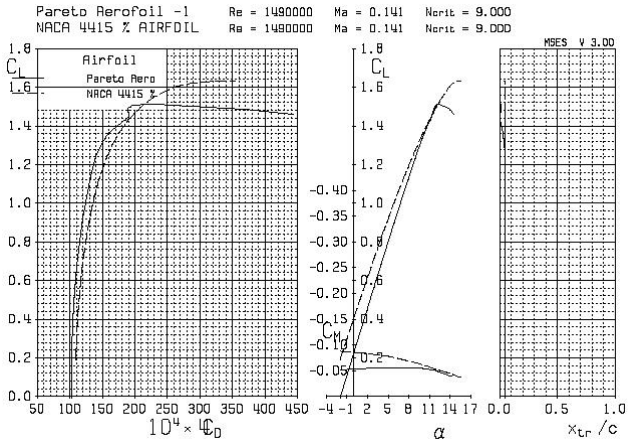


Figure 26: Comparative Polars – Compromise Aerofoil and NACA 4415 *Re=1.490 x 10^6* - c_m Constrained

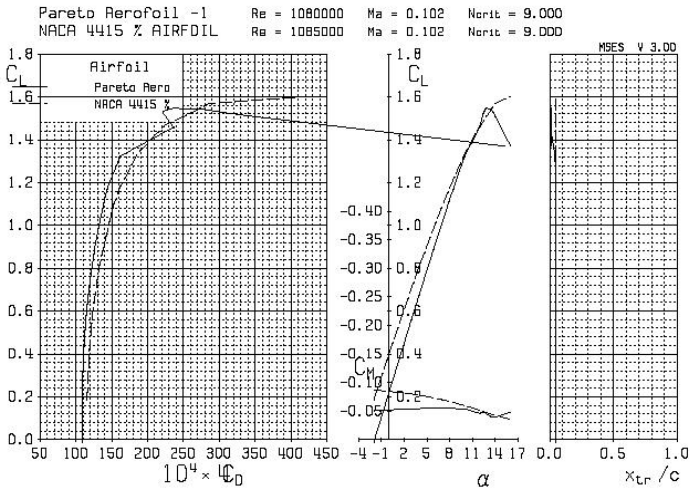


Figure 27: Comparative Polars – Compromise Aerofoil and NACA 4415 *Re=1.085 x 10^6* - c_m Constrained

Concluding this case, it is apparent that the evolved aero foils offer significantly lower drag at both cruise conditions. The requirement of constraining the pitching moment during the evolution

process is necessary to avoid obtaining an aerofoil with lower drag for some flight conditions but with undesirable pitching moment characteristics. The results obtained show the capabilities of the method to find optimal solutions and classical aerodynamic shapes for flow drag. The importance of sound engineering judgement before, during and after the optimisation cannot be under-emphasized; a proper definitions of constraints before performing the evolutionary optimisation and the final results need to be evaluated to obtain feasible designs.

Description	Flight Condition One —Slow Cruise	Flight Condition Two —Fast Cruise
NACA 4415	0.01621	0.01173
Pareto 01 — c_m Constrained	0.01451 [-10.48 %]	0.01090 [-7.07 %]

Table 2: UAV Drag reduction at two operating conditions.

Concluding these test cases, it is shown that, using the evolutionary approach and methods developed, it is possible to capture the Nash equilibrium and complex Pareto fronts describing the trade-off between the objectives for direct and inverse problems. The main advantage of the method is that it can find globally optimum Pareto fronts and that it is integrable with a pre-existing flow-solver without modification or differentiation. The benefit of parallel computation is also clear.

9. CONCLUSIONS

These notes have described the characteristics and implementation details of a framework for design and optimisation of aeronautical systems. Numerical details indicated that several requirements on problem formulation, execution need to be taken under consideration. Details on algorithms and modern extensions were provided. Several examples of mathematical and aerodynamics test cases to numerically capture Pareto fronts for inverse design, and Nash equilibrium were described step by step. From the results obtained on a parallel cluster of PCs it is shown that the method and evolutionary algorithms have good parallel properties and perform well with an increasing number of processors.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge S. Armfield, University of Sydney for access to the PC cluster of computers. We would like to thank Arnie McCullers at NASA LaRC who kindly provided the *FLOPS* software and M. Drela at MIT for providing the *MSES* software.

REFERENCES

1. Abott, H and Von Doenhoff, A. E. Theory of Wing Sections, Dover, 1980.
2. Alexandrov, N. M. and Lewis, E. M., Comparative Properties of Collaborative Optimisation and Other Approaches to MDO, Proceedings of the First ASMO UK / ISSMO Conference on Engineering Design Optimization, July 8-9, La Jolla, California, July 8-9, 1999.
3. Alexandrov, N. M. and Lewis, R. M., Analytical and Computational Properties of Distributed Approaches to MDO, AIAA 2000-4718, September 2000.
4. Bartholomew, P. The Role of MDO within Aerospace Design and Progress towards an MDO Capability, AIAA-98-4705, pp 2157-2165, 7th AIAA/USAF/NASA/ ISSMO Symposium on Multidisciplinary Analysis and Optimization, AIAA, St. Louis, Mo,1998.
5. Braum, R., Gage, P., Kroo, I. and Sobieski , I. Implementation and Performance Issues in C.O, NASA-AIAA-96-4017, 1996.
6. Cantu-Paz, E. Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Pub, 2000.

7. Coello Coello, C.A., Van Veldhuizen, D.A. and Lamont, G.B. Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic Publishers, New York, March, 2002.
8. Collard, P. and Escazut, C. Genetic Operators in a Dual Genetic Algorithm, In Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence, Virginia, USA,. IEE pp. 12-19, November 1995.
9. Deb, K. Multi-Objective Optimization Using Evolutionary Algorithms, Wiley, 2003.
10. Drela, M. XFOIL 6.94 User Guide. MIT Aero Astro, 2001.
11. Drela, M. A User's Guide to MSES V2.3, Feb. 1993.
12. Geist, A., Beguelin, A., Dongarra, J. , Jiang, W. , Manchek, R. and Sunderam, V. PVM: Parallel Virtual Machine. A User' s Guide and Tutorial for Networked Parallel Computing. Massachusetts Institute of Technology, 1994
13. Goldberg, D. Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley,1989.
14. González, L.F, Whitney, E., Srinivas, K. and Periaux, J. Multidisciplinary Aircraft Design and Optimisation Using a Robust Evolutionary Technique with Variable Fidelity Models. AIAA Paper 2004-4625, In CD Proceedings 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Aug. 30 - Sep. 1, 2004, Albany, NY.
15. Gonzalez, LF, Whitney, E.J. , Periaux, J., Sefrioui, M. and Srinivas, K. A Robust Evolutionary Technique for Inverse Aerodynamic Design, Design and Control of Aerospace Systems Using Tools from Nature; Proceedings of the 4th European Congress on Computational Methods in Applied Sciences and Engineering, Volume II, ECCOMAS 2004, Jyvaskyla, Finland, July 24-28, 2004, Eds: P. Neittaanmaki,T. Rossi, S. Korotov , E. Onate, J. Periaux and D. Knorzer, University of Jyvaskyla, Jyvaskyla, CD ISBN 951-39-1869-6.
16. Gonzalez, L. F., Whitney, E. J., Srinivas, K., Wong, K. C. and Périaux , J. Optimum multidisciplinary and multi-objective wing design in cfd using evolutionary techniques, Computational Fluid Dynamics 2004, Springer Berlin Heidelberg, 681-686.
17. Lee. DS, Gonzalez, L. F., Srinivas, K., Auld, D. and Périaux , J. Multi-Objective / Multidisciplinary Design Optimisation of Blended Wing Body UAV via Advanced Evolutionary Algorithms AIAA Paper 2007–36, 45th ,45th AIAA Aerospace Sciences Meeting and Exhibit. January 2007
18. Hansen, N. and Ostermeier, A. Completely Derandomised Self-Adaption in Evolution Strategies. In *Evolutionary Computation*, volume 9(2), pages 159–195. MIT Press, 2001.
19. Holland, J.H. Adaption in Natural and Artificial Systems. University of Michigan Press. 1975.
20. Jameson, A., Caughey, D., Newman, P. and Davis, R. .A Brief Description of the Jameson Caughey NYU Transonic Swept-Wing Computer Program FLO22, NASA Technical Memorandum, NASA TM X-73996, Dec. 1976.
21. Jameson, A. A Perspective on Computational Algorithms for Aerodynamic Analysis and Design. *Progress in Aerospace Sciences*, 37(2):197–243(47), February 2001.
22. Kim, H. J. and Rho, O. H.. Dual-Point Design of Transonic Airfoils Using the Hybrid Inverse Optimization Method. *Journal of Aircraft*, 34(5):612–618, September–October 1997.
23. Kroo, I., Altus, S., Braun, R., Gage, P. and Sobieski, I., Multi disciplinary Optimization Methods for Aircraft Preliminary Design, AIAA 94-4325, Fifth AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, September 7-9, Panama City, Florida, 1994.
24. McCullers, A. FLOPS User's Guide, Release 6.02, NASA Langley Research Center, March 2003.

25. Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs. Artificial Intelligence. Springer-Verlag, 1992.
26. Mohammadi, B. Fluid Dynamics Computation with NSC2KE: A User-Guide: Release 1.0. Rt-0164, INRIA, 1994.
27. Obayashi, S. Multidisciplinary Design Optimization of Aircraft Wing Planform Based on Evolutionary Algorithms. In Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics, La Jolla, California, IEEE, October 1998.
28. Obayashi, S. and Takanashi, S. Genetic Optimization of Target Pressure Distributions for Inverse Design Methods. *AIAA Journal*, 34(5):881–886, May 1996.
29. Oyama, A., Liou, M.S. and Obayashi, S. Transonic Axial-Flow Blade Shape Optimization Using Evolutionary Algorithm and Three-Dimensional Navier-Stokes Solver, 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, Georgia, September, 2002.
30. Périaux, J., Lee, DS., Gonzalez, LF., Srinivas, K. Fast reconstruction of aerodynamic shapes using evolutionary algorithms and virtual nash strategies in a CFD design environment, Journal of computational and applied mathematics, 2009
31. Press, W., Teukolsky, S., Vetterling,W. and Flannery, B. *Numerical Recipes in C: The Art of Scentific Computing*. Cambridge University Press, second edition,1992.
32. Raymer, D. Aircraft Design: A Conceptual Approach, American Institute of Aeronautics and Astronautics American Institute of Aeronautics and Astronautics, Third Edition, 1999.
33. Salas, A. O. and Townsend, J. C. Framework Requirements for MDO Application Development, 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, Missouri, AIAA 98-4740, September 2-4, 1998, pp. 11.
34. Sankar, M. R., Isaacs, A., Mujumdar, PM., Sudhaka, K., MDO Framework Development - A Case Study With An Elementary Model Of Airborne Early Warning System Optimization, 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, Georgia, September 4-6, 2002
35. Sefrioui, M. and Périaux, J. A Hierarchical Genetic Algorithm Using Multiple Models for Optimization. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo and H.-P. Schwefel, editors, Parallel Problem Solving from Nature, PPSN VI, pages 879-888, Springer, 2000.
36. Sobieski, J., Haftka, R. Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments, AIAA Paper No. 96-0711, 1996.
37. Sobieczky, H, Parametric Airfoils and Wings, *Recent Development of Aerodynamic Design Methodologies – Inverse Design and Optimisation*-, Friedr. Vieweg & Sohn Verlagsgesllschaft mbH, Braunschweig/Wiesbaden, pp.72-74,1999,
38. Thierens, D. Adaptive mutation rate control schemes in genetic algorithms.. In (Ed.), Proceedings of the 2002 IEEE World Congress on Computational Intelligence: Congress on Evolutionary Computation (pp. 980-985). IEEE Press, 2002.
39. Thomas, Z. and Green, A. Multidisciplinary Design Optimization Techniques: Implications and Opportunities for Fluid Dynamics Research. AIAA Paper-1999-3798, Jun, 1999.
40. Van Veldhuizen, D. A., Zydallis, J.B. and Lamont, G. B. Considerations in Engineering Parallel Multi objective Evolutionary Algorithms, IEEE Transactions on Evolutionary Computation, Vol. 7, No. 2, pp. 144--173, April 2003.
41. Wakunda, J., Zell, A. Median-selection for parallel steady-state evolution strategies. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, Parallel Problem Solving from Nature – PPSN VI, pages 405–414, Berlin, Springer, 2000.
42. Whitney, E. J. A Modern Evolutionary Technique for Design and Optimisation in Aeronautics. PhD Thesis, The University of Sydney, 2003.

43. Whitney, E., Sefrioui, M., Srinivas, K. and Périaux, J.: Advances in Hierarchical, Parallel Evolutionary Algorithms for Aerodynamic Shape Optimisation, JSME (Japan Society of Mechanical Engineers) International Journal, Vol. 45, No. 1, 2002.